MEMORANDUM M-842

Lagrange-like contact in ETA
+
User's Manual

P.T.G. Volgers

March, 1998

Delft University of Technology
Faculty of Aerospace Engineering
Delft, The Netherlands
Lagrange-like contact in ETA

P.T.G. Volgers

Master’s Thesis
January 1997
Thesis advisor: dr. ir. E. Riks
Abstract

This report describes the theory and implementation of a contact algorithm in the explicit finite element code ETA.

First a motivation of the choice of the method is given. Then the theory of this method is described.

The new input definition and the new data sets are described. Implementation is given, following the different subroutines.

Tests and demos are shown to validate the algorithm. Finally results are discussed and recommendations made concerning future developments.
Thesis Committee

prof. dr. J. Arbocz

dr. ir. E. Riks

dr. ir. A. de Boer

ir. R.W. Fransen
Preface

In 1995 I worked as a trainee at Engineering Systems International (ESI) France, where I developed a new contact segment based on a higher order description of the surface. After returning to start on my thesis work, I was asked by dr. Riks to continue on the subject of contact in finite elements and to develop and implement a contact-impact algorithm in a finite element program.

This thesis work consists of two parts. The first report is an extensive literature study to describe the different solution methods available today, for both implicit and explicit finite element methods. The second part describes the implementation of one of these methods in the explicit finite element program ETA. This program was first developed by Rogier Fransen for his master's thesis. It is based on the database management system MEM-COM and uses the same data structure as the implicit finite element program B2000.

This report describes the theory and the implementation of the contact in the finite element code ETA. This includes the modifications to the input processor and a description of the new subroutines.

This report contains 87 pages.

P.T.G. Volgers,

Delft, November 1996
Acknowledgements

I would like to thank the following people for their contributions, remarks and other assistance during my thesis work:

First of all dr. E. Riks at the faculty of Aerospace Engineering for his supervision, his assistance, the useful discussions (not only on mechanics) and for letting me do it my way. And prof. dr. J. Arbocz also for his supervision.

The people at ESI for gaining experience with explicit codes and contact problems. And especially ir. Rogier Fransen, being my predecessor on developing ETA and my direct supervisor at ESI, for helping me to learn about contact-impact problems and for comments on my thesis report.

And dr. Silvio Merazzi from SMR, Switzerland, dr. Andre de Boer from the NLR and ir. Gert Rebel from the faculty for their help with MEM-COM and B2000.
Contents

1 Introduction 1

2 The contact-impact problem 3
  2.1 Definition of the contact surface 3
  2.2 The contact search 4
  2.3 The contact conditions 5

3 The contact definition 11
  3.1 Input commands 11
  3.2 Data sets 13

4 Implementation 15
  4.1 The contact search routine 16
  4.2 The contact force routine 19

5 Test results 23
  5.1 Normal contact force 24
    5.1.1 Plate pressed on master segment 24
    5.1.2 Plate falling on master segment 24
    5.1.3 Beam on rigid wall 25
  5.2 Friction 30
  5.3 Ball impacting on plate 31
  5.4 A sample stamping problem 32
  5.5 Two tubes impacting 34

6 Conclusions and recommendations 37

Bibliography 39

A Source files 41
  A.1 b2ipcontact.F 41
  A.2 etacontact.F 49
| A.3 | etafind01.F | 53 |
| A.4 | etacon01.F | 59 |

<table>
<thead>
<tr>
<th>B</th>
<th>Input files test problems</th>
<th>65</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.1</td>
<td>contact1.i</td>
<td>65</td>
</tr>
<tr>
<td>B.2</td>
<td>contact2.i</td>
<td>68</td>
</tr>
<tr>
<td>B.3</td>
<td>friction.i</td>
<td>70</td>
</tr>
<tr>
<td>B.4</td>
<td>beam.i</td>
<td>73</td>
</tr>
<tr>
<td>B.5</td>
<td>beam.inp</td>
<td>76</td>
</tr>
</tbody>
</table>
List of Figures

2.1 The hierarchical contact system ............................................. 4
2.2 Bounding box for a 3D contact segment .................................... 4
2.3 Hitting node and defence node ............................................. 6
2.4 Forces on a contact pair .................................................. 8

5.1 Two contacting plates, top view ........................................... 23
5.2 External force and contact force .......................................... 24
5.3 Displacement of the falling plate ......................................... 25
5.4 Beam against rigid wall, model ........................................... 25
5.5 Beam against rigid wall, contact force with Lagrange method.
     \( L_1 = \text{ETA}, \text{ scalefactor} = 0.8; \ L_2 = \text{analytical solution}. \) 26
5.6 Beam against rigid wall, contact force with Lagrange method.
     \( L_1 = \text{ETA}, \text{ scalefactor} = 0.08; \ L_2 = \text{analytical solution}. \) 28
5.7 Beam against rigid wall, contact force with Lagrange method.
     \( L_1 = \text{ABAQUS using 21 steps}; \ L_2 = \text{ABAQUS using 54 steps}; \)
     \( L_3 = \text{analytical solution}. \) ........................................... 28
5.8 Beam against rigid wall, contact force with penalty method in
     ETA. \( L_1 = \text{analytical solution}; \ L_2 = \text{penalty parameter} \ \kappa = 1e6 \) 29
5.9 Beam against rigid wall, contact force with penalty method
     in ETA. \( L_1 = \text{analytical solution}; \ L_2 = \text{penalty parameter} \ \kappa = 1e5; \)
     \( L_3 = \text{penalty parameter} \ \kappa = 1e4. \) ........................................... 29
5.10 Plate sliding with friction, top view .................................... 30
5.11 Sliding of a plate, tangential velocity .................................. 30
5.12 Ball on plate ............................................................. 31
5.13 Ball on elastic plate, maximum plate deflection ........................ 31
5.14 Ball falling on two plates, multiple contact ............................ 32
5.15 Stamping problem, model ............................................... 33
5.16 Stamping problem, deformed configuration ................................ 33
5.17 Two impacting tubes, model ............................................. 34
5.18 Two impacting tubes, full collapse ...................................... 35
5.19 Two impacting tubes, maximum deformation due to element
     instability ................................................................. 36
List of Symbols

\begin{itemize}
\item \textit{a} \quad \text{Acceleration field}
\item \textit{b} \quad \text{Body forces}
\item \textit{E} \quad \text{Young's modulus}
\item \textit{f, F, R} \quad \text{Force vector}
\item \textit{G} \quad \text{Shear modulus}
\item \textit{m} \quad \text{Mass}
\item \textit{N} \quad \text{Unit outward normal}
\item \textit{q} \quad \text{Surface traction}
\item \textit{s, s_{ij}} \quad \text{Second Piola-Kirchhoff stress tensor}
\item \textit{t} \quad \text{Time}
\item \textit{u} \quad \text{Displacement field}
\item \textit{v} \quad \text{Velocity field}
\item \textit{x} \quad \text{Spatial coordinates}
\item \textit{\epsilon, \epsilon_{ij}} \quad \text{Strain tensor}
\item \textit{\kappa} \quad \text{Penalty parameter}
\item \textit{\lambda} \quad \text{Lagrange multiplier, scalar}
\item \textit{\nu} \quad \text{Coulomb friction coefficient or Poisson's coefficient}
\item \textit{\rho} \quad \text{Density}
\item \textit{\sigma, \sigma_{ij}} \quad \text{Cauchy stress tensor}
\item \textit{\phi_i} \quad \text{Shape function}
\item \textit{\langle \cdot \rangle} \quad \text{Macauley brackets}
\end{itemize}
Chapter 1

Introduction

In 1994 Rogier Fransen wrote the explicit finite element code ETA (which stands for *Explicit Transient Analysis*) build on the database management system MEM-COM and using the same data structure of the implicit code B2000. It was developed, as the name already indicates, for transient structural analysis, such as vibrations, stamping and crash. The code had only the very basic tools necessary for the analysis. It consisted of only one shell element and a linear elastic isotropic material model. In his thesis report [1, 2] several recommendations were made for future developments. One of these was the implementation of a contact algorithm.

After an extensive literature study of the different methods for contact-impact models in a finite element code [9] a suitable method was chosen to be implemented. For a complete description of the different methods one is referred to this report. To calculate the contact forces two basic methods are available: the first is the Lagrange multiplier method, which calculates the unknown contact forces by introducing them as multipliers and enforces the non-penetration condition to calculate this multipliers. The second is the penalty method, which allows for penetration of one body into the other and uses this penetration to calculate the contact force by penalizing this penetration. The penalty factor can be seen as the stiffness of a linear spring for the penetration. Selection of the method to be implemented was based on the following considerations:

- First (and most) of all the algorithm has to be suitable for an *explicit* finite element code. This means that the equations for the contact algorithm have to be decoupled and do not require an iterative solution procedure. This excludes the direct implementation of the Lagrange multiplier method, as this incorporates a coupled set of equations.

- Second, the method has to be flexible enough to be applied to *general*
problems. This means that the method must be a general node-to-segment contact algorithm and not a method were one has to know in advance which node comes in contact with which node.

- Third, the method will be used by users who are not experts on contact-impact solution methods. Although the demands mentioned above direct to the use of the penalty method, it is the experience of the author that the influence of the penalty method is usually not thoroughly investigated. Also the penalty method reduces the critical time step of the explicit code with increasing penalty parameter.

Taking all these considerations into account, the choice was made to implement a Lagrange-like contact type, the so-called *defence node algorithm* developed by Zhong [10, 11, 12]. This algorithm decouples the equations and still enforces the non-penetration condition. Also this method does not require the use of a penalty parameter, making it easier to use.

This report will first give a brief summary of the theory of the contact-impact equations and the contact search focused to the methods used in ETA. Then the new input in the input file and processing in the input processor is described (Chapter 3), followed by the implementation in the ETA code (Chapter 4). In Chapter 5 test results are shown and in the last chapter conclusions are drawn and recommendations are made.
Chapter 2

The contact-impact problem

In the first report [9] the theory and solution methods for contact-impact in finite elements (FE) is described. This includes the strong and weak formulation as well as the different solution methods in use today. Also different methods applied to contact search can be found in that report. In this report however, we will concentrate on the methods as they are implemented in the explicit FE code ETA. Also the order of discussing the different aspects of the contact-impact problem will be different, as we follow the implementation of the theory in the FE code. This means, that first we will describe the contact surface definition, followed by the contact search. Then finally the calculation of the contact forces is described, including not only the normal contact force, but also the treatment of friction and contact damping.

2.1 Definition of the contact surface

To define the contact surfaces that can come into contact, we use a hierarchy based system, as shown in Fig.(2.1) and discussed in [9], Ch.7. The general contact system consists of different contact bodies. These are usually different bodies, but they can also be different parts of one structure. Of each contact body we can define one or more contact surfaces, i.e. that part (or parts) of the body where contact might occur. Each contact surface is made out of contact segments, the 'elements' of the contact surface. And like the structural elements, the contact segment is defined by its contact nodes.

The contact solution method for the FE method is a so-called node-to-segment contact. This means that the contact conditions are not imposed on one surface to another, but to a contacting node, called a slave node, on a contact segment, the master segment. Thus, for every contact surface we define the contact segments (the master segments) and a set of nodes which
might come into contact with this surface (the slave nodes). The contact-impact problem is now reduced to finding which slave node comes into contact with which master segment and to impose the contact conditions.

2.2 The contact search

Now that we have defined the master segments and the possible contacting slave nodes, we have to find a way to determine the nodes in contact. The simplest method would be to calculate for each slave node the distance to each master segment. If the distance is zero, the slave node is in contact with that particular master segment. Of course, this is a computationally very expensive method. To improve the contact search we divide the contact search for each master segment into a global search and a local search.

To perform the global search we define a search box around the segment, as shown in Fig.(2.2). The boundaries of the box (the dashed lines) are given by the maximum and minimum values for the coordinates x, y and z. The possible contacting nodes are now to be found within this bounding box. We now extend the bounding box to all sides with a factor h (the dotted
lines) to include all nodes which are in the vicinity of the contact segment. A slave node becomes a possible contacting node if it satisfies the following conditions:

\[ x_{\text{min}} - h \leq x^s \leq x_{\text{max}} + h \]  
\[ y_{\text{min}} - h \leq y^s \leq y_{\text{max}} + h \]  
\[ z_{\text{min}} - h \leq z^s \leq z_{\text{max}} + h \]

where \( x^s \) the x-coordinate of the slave node, \( x_{\text{min}} \) the minimum x-coordinate of the master segment, etc. Because the displacements per time step are relatively small, we do not have to perform the global contact search every cycle. The bigger the global search box, the more cycles we do not have to perform the global search. However, the more nodes in the box, the more computations we have to make for the local search.

If the contact segment and/or the slave nodes are part of a shell element, the segment and/or slave nodes can have a certain thickness, defined by the contact thickness. This contact thickness also forces the bounding box to expand.

The next step will now be the local contact search. For each possible contacting node its projection on the master segment is calculated. If the projection is not on the master segment, the slave node cannot be in contact with the master segment. Otherwise, the distance \( d \) to the master is calculated according to:

\[ d = (x^t - x^s) \cdot N_1 - t \]

where \( N_1 \) the unit outward normal of the master segment at the contact point, \( t \) the contact thickness (for example half of the thickness of the shell element) and \( x^t \) and \( x^s \) the coordinates of the target node and slave node respectively. The penetration \( p \) of the slave node into the master segment is given by

\[ p = \langle -d \rangle \]

where \( \langle \cdot \rangle \) the so-called Macauley brackets, defined as \( \langle x \rangle = 1/2(x + |x|) \) This penetration \( p \) is used to calculate the contact force.

### 2.3 The contact conditions

To calculate the contact force and to impose the contact conditions, we have two basic methods: the Lagrange multiplier method and the penalty method.
The penalty method is the most widely used method as it uses a penalization to calculate the normal contact force directly:

\[ F_N^s = (\kappa \cdot p)N_1 \quad (2.6) \]

where \( \kappa \) a penalty parameter, which can be seen as a kind of 'contact stiffness'. The Lagrange multiplier method however enforces the non-penetration condition (or kinematic contact condition):

\[ p = 0 \quad (2.7) \]

and introduces the contact force by means of an unknown multiplier. In ETA this method is implemented as follows [9, 11].

If a slave node comes into contact with a master segment, a so-called defence node is constructed at the target point on the master segment, as shown in Fig.(2.3). This defence node is the representation of the master segment for the slave node and has all the characteristics of a contact node. Its function is to prevent the slave node to penetrate into the master segment and forms a contact pair with the slave node. The velocity \( \mathbf{v} \) and acceleration \( \mathbf{a} \) of the defence node are given by

\[ \mathbf{v}^d = \sum_{n=1}^{N} \phi_n \mathbf{v}^n \quad (2.8) \]

\[ \mathbf{a}^d = \sum_{n=1}^{N} \phi_n \mathbf{a}^n \quad (2.9) \]

where \( (\cdot)^d \) denotes the defence node, \( (\cdot)^n \) denotes the target nodes of the corresponding master segment and \( \phi^n \) the corresponding shape function. If the defence node has a mass \( m^d \) and if \( \mathbf{R}^d \) the sum of internal and external forces, except the contact force, acting on this node and if \( \mathbf{f} \) the contact force vector between the nodes of the contact pair, then the equation of motion of the defence node can be written as:

\[ m^d \mathbf{a}^d = \mathbf{R}^d + \mathbf{f} \quad (2.10) \]
This contact force $f$ is the unknown force to ensure the non-penetration condition. This force is then distributed over the nodes of the master segment, so that the equation of motion for the master nodes becomes:

$$m^d a^m = R^n + f^n$$  \hfill (2.11)

Combining Eqs.(2.10) and (2.11) yields:

$$m^d \sum_{n=1}^N \phi_n \frac{R^n + f^n}{m^n} = R^d + f$$  \hfill (2.12)

It is obvious that the resulting force vector on the defence node $R^d$ depends only on $R^n$ and the contact force on the master nodes depends only on the contact force of the defence node, so that we can write:

$$m^d \sum_{n=1}^N \phi_n \frac{R^n}{m^n} = R^d$$  \hfill (2.13)

$$m^d \sum_{n=1}^N \phi_n \frac{f^n}{m^n} = f$$  \hfill (2.14)

Using Eq.(2.13) we can compute $R^d$ directly, but to calculate $f^n$ from $f$ we have to assume that

$$f^n = \Phi^n f \quad n = 1, N$$  \hfill (2.15)

where $\Phi^n$ some function which must be chosen. As the contact force $f$ is distributed over the nodes to calculate $f^n$, it is obvious that

$$\sum_{n=1}^N f^n = f$$  \hfill (2.16)

so that

$$\sum_{n=1}^N \Phi^n = 1$$  \hfill (2.17)

Now we have to determine $\Phi^n$. If we want the defence node to have the exact same characteristics as a master node when the slave node is in contact with that particular master node, we obtain for $\Phi^n$ [9]:

$$\Phi^n = \frac{m^n \phi_n}{m^d \phi_2}$$  \hfill (2.18)
\[ F_n^d \rightarrow \bullet \rightarrow f_n^d \quad F_n^h \rightarrow \bullet \rightarrow f_n^h \]

Figure 2.4: Forces on a contact pair

where

\[ \bar{\phi}_2 = \sum_{n=1}^{N} (\phi_n)^2 \quad (2.19) \]

If more than one slave node comes in contact with this master segment or one of its adjacent master segments, the force vectors \( \mathbf{R} \) include the contribution of the other contact forces, unknown either. This way, we obtain a coupled system of equations. To decouple these equations in order to preserve the efficiency of the explicit method, we neglect at \( t = t_i \) the contact forces at time \( t_i \), but use the contact forces at \( t = t_{i-1} \) instead. The error we now introduce will be corrected in the next time step. Now the vectors \( \mathbf{R} \) are known and the system is decoupled.

To calculate the unknown contact force by imposing the non-penetration condition, we now only have to consider one contact pair: a hitting slave node \( h \) and the corresponding defence node \( d \), see Fig. (2.4). The equations of motion for the slave node and the defence node in the normal direction become:

\[ m^\alpha a^\alpha = F_n^\alpha + f_i^\alpha \quad \alpha = d, h \quad (2.20) \]

Integration of Eq.(2.20) with the predictor-corrector method yields:

\[ \frac{m^\alpha}{\Delta t_i} \left[ \frac{u_{i+1}^\alpha - u_i^\alpha}{\Delta t_i} - v_{i-1/2}^\alpha \right] = F_i^\alpha + f_i^\alpha \quad \alpha = d, h \quad (2.21) \]

The kinematic contact condition becomes:

\[ p_{i+1} = p_i + \left( u_i^d - u_{i+1}^d \right) - \left( u_i^h - u_{i+1}^h \right) = 0 \quad (2.22) \]

Combination of Eq.(2.21) and Eq.(2.22) gives us the expression for the contact force:

\[ f_i^h = -f_i^d = \frac{m_i^h m_i^d}{m_i^h + m_i^d} \left[ \frac{F_i^d}{m_i^d} + \frac{F_i^h}{m_i^h} \right] \left[ \frac{v_{i-1/2}^d}{\Delta t_i} - \frac{v_{i-1/2}^h}{\Delta t_i} + \frac{P_i}{\Delta t_i^2} \right] \quad (2.23) \]

This force is directed along the unit outward normal.
To describe friction, the classical Coulomb friction law is implemented. Of course other friction models are available, but the Coulomb friction is the basic and simplest friction model and can also be implemented and validated with relative ease. Implementation is as follows: in case of contact between a slave node and its defence node, the contact is assumed to be sticking, i.e. no relative sliding will occur.

\[ \Delta v_t = 0 \] (2.24)

where the subscript \( t \) denotes 'in tangential direction'. The friction force to enforce this condition is calculated and compared with the maximum static contact force.

\[ f_{fric}^h = \frac{m^h m^d}{m^h + m^d} \left[ \frac{F_t^d}{m^d} - \frac{F_t^h}{m^h} + \frac{v_{t-1/2}^d}{\Delta t_t} - \frac{v_{t-1/2}^h}{\Delta t_t} \right] \] (2.25)

\[ f_{max} = \nu f_n \] (2.26)

where the friction coefficient \( \nu \) can be either the static friction coefficient \( \nu_s \) or the dynamic friction coefficient \( \nu_d \) and \( f_n \) the magnitude of the contact force in normal direction. If the friction force is greater then or equal to the maximum friction force (i.e. the static or dynamic friction force, depending whether the contact is sticking or sliding), the friction is limited to the dynamic friction force and directed opposite to the relative sliding direction.

\[ f_{fric} = -\nu_d f_n \left( \frac{\Delta v_t}{|\Delta v_t|} \right) \] (2.27)

where \( |a| \) denotes the length of vector \( a \) and \( \nu_d \) the dynamic friction coefficient, \( \nu_d \leq \nu_s \).

Contact damping is also included by means of a user-defined damping coefficient. In practice the choice of the correct damping parameter turns out to be rather awkward, so implementation to use the critical damping coefficient is recommended, but this is not as straightforward as by using the penalty parameter (see [9], Ch.6).

The contact damping is calculated as follows:

\[ f^h = \langle f^h + c \cdot \Delta v_n \rangle \] (2.28)

where \( c \) the contact damping parameter, \( \Delta v_n \) the relative velocity in normal direction and \( \langle \cdot \rangle \) denotes the Macauley brackets, defined as \( \langle x \rangle = 1/2(x+|x|) \).
Chapter 3

The contact definition

In the finite element program B2000/ETA [4, 6] the model definition is written in an input file. This file is read and interpreted by the input processor b2ipeta.x (IP). The IP then stores all necessary data in a database, controlled by the database management system MEM-COM [5]. The calculation processor b2eta.x (ETA) reads and writes all information from and to this database. First the command language for the input file is described. Then the storage in the database and the new data sets are described.

3.1 Input commands

The definition of the contact interfaces is straightforward. The use of the B2000 input processor and input syntax provide a command driven definition of the contact interface. These and other input definitions can also be found in the manual [6, 8]. A complete contact definition using default values is written as:

```plaintext
! CONTACT
   CSID  1
   CTYPE 1
   MTHICK  0.0
   STHICK  0.0
   DAMPING  0.0
   SFRICITION  0.0
   DFRICITION  0.0
   GSEARCH  21
   DIRFLAG  1
```
MASTER
1 1 2 3 4
2 5 6 7 8
SLAVE
9/15/1
ENDC
END

The contact definition starts with the command CONTACT (uppercase or lowercase) and ends with the command END.

Then the contact surfaces are defined. Always the first command of a contact surface definition is the contact surface ID number, CSID. Contact surfaces must be numbered successively. The maximum number of contact surfaces to be defined is set to 100. The contact surface definition ends with the command ENDC, whereafter a new surface can be defined or the contact definition is ended. The order of the following commands is at will and they may be abbreviated to 4 characters. If a parameter is defined twice, only the last definition is used.

CTYPE defines the contact type number: contact type number 1 is a general node-to-segment contact, contact type number 2 is for rigid foundations with all displacements prescribed. This reduces CPU-time compared to type 1. Contact type 3 is only partially completed. It uses the penalty method to calculate the contact force, but the penalty parameter $\kappa$ is fixed and no friction is included. This contact type should not be used. No default value is assumed.

THICK, MTHICK defines the thickness of the master segment (e.g. the half thickness of the shell elements on which the contact surface is defined) and STHICK the contact thickness of the slave surface. The default value is zero for both surfaces.

DAMPING defines the structural damping coefficient. As already mentioned before, this cannot be done by providing a critical damping value. Default value is zero.

FRICITION, SFRICTION and DFRICTION define the different friction coefficients. If only FRICITION is given, the static and dynamic friction coefficient are equal. Otherwise, both must be given. The default values are all zero.

GSEARCH is the global search counter. Global search is performed every GSEARCH cycles. The default value is set to 21.

DIRFLAG defines the direction of the unit outward normal of the master segment. The default takes the master segment nodes counter-clockwise. DIR -1 sets the other side of the segment as the outside. Default is 1.
MASTER is the command to define the master segments of the contact surface, syntax is: segment, node1, node2, node3, node4. The segments are (like elements) defined by nodal numbers, but they do not have to coincide with the structural elements. Master segments of contact type 1 and 2 are always made of 4 nodes. For contact type 1 the nodes must be connected to structural elements, for contact type 2 all the displacements must be prescribed. The segment numbers can be chosen randomly, only internal segment numbers are stored. The maximum number of master segments per contact surface definition is set to 5000, no default assumed.

SLAVE defines the slave nodes to the master segment according to the following syntax: firstnode/lastnode/nodestep. Slave nodes cannot be part of the master definition. The maximum number of slave nodes per contact surface definition is set to 10000, no default assumed.

3.2 Data sets

To store the contact definition data, some new data sets are defined. For more information about the MEM-COM database management system, see [4, 5]. The data set CDIR.1 contains the global data for all contact surfaces:

- TOTSURF is the total number of contact surface definitions.
- TOTMASTER is the total number of master segments of all contact surfaces.
- TOTSLAVE is the total number of slave nodes of all contact surfaces.

Then for each contact surface definition, the data is stored in a separate data set, named CONT.1.CSID, where CSID the contact surface number. This data set contains the following data:

- CTYPE the contact type number.
- GSC the global search counter.
- NMASTER the total number of master segments.
- MASTER the set of master segment definitions.
- NSLAVE the total number of slave nodes.
- SLAVE the slave node numbers.
- MTHICK and STHICK the contact thickness of the master and the slave segments.

- SFRICITION and DFRICITION the static and dynamic friction coefficients.

- DAMPING the contact damping coefficient.

All the data is read by the control module of ETA, b2etacm, to be used for the calculations. This control module first reads the data set with the global contact data CDIR.1 and uses this information to allocate the arrays correctly. The subroutine etacontinit initializes the arrays with information from the data sets CONT.1.CSID. The following arrays contain the contact data, where isurf denotes the contact surface number, inode the global nodal number and islave the slave node numbers:

- csurface(1,isurf) contact type
- csurface(2,isurf) number of master segments
- csurface(3,isurf) number of slave nodes
- csurface(4,isurf) number of cycles per global search
- csurface(5,isurf) global search counter, set to zero

- cdata(1,isurf) contact thickness master segments
- cdata(2,isurf) contact thickness slave segments
- cdata(3,isurf) static friction coefficient
- cdata(4,isurf) dynamic friction coefficient
- cdata(5,isurf) damping coefficient

- master( ) master surface definition
- slave(islave) slave node numbers
- invslave(inode,isurf) inverse of slave( )

The array slave(islave) returns the global node number inode for slave node number islave while the array invslave(inode,isurf) returns the slave node number islave of the contact surface isurf for the global node number inode. The array invslave is therefore the inverse of the array slave.
Chapter 4

Implementation

In this chapter the implemented subroutines are described. The subroutines can also be found in Appendix A.

In case contact is included in the problem to be solved, the contact routine etacontact is the first routine in the time integration cycle. In the contact routine a loop is made over all the contact surface definitions.

\[
\text{do 100 isurf=1,TUTSURFACE}
\]
\[
\text{ctyp=csurface(1,isurf)}
\]
\[
\text{nmaster=csurface(2,isurf)}
\]
\[
\text{nslave=csurface(3,isurf)}
\]
\[
\text{call b2setint(0,contactflag,MAXSLAVE)}
\]

In this loop first the contact data for the surface is retrieved and the contactflag, indicating whether a slave node is in contact or not, is set to zero for all nodes. Then, depending on the contact type, a loop is made over all the master segments.

\[
\text{if (ctyp.eq.1) then}
\]
\[
\text{do 10 imaster=1,nmaster}
\]
\[
\text{call etafind01(Xold ,csurface(4,isurf), ...)}
\]
\[
\text{call etacont01(Xold ,forc , ...)}
\]
\[
\text{endif}
\]
\[
\text{if (ctyp.eq.2) then}
\]
\[
\text{do 20 imaster=1,nmaster}
\]
\[
\text{call etafind01(Xold ,csurface(4,isurf), ...)}
\]
\[
\text{call etacont02(Xold ,forc , ...)}
\]
\[
\text{continue}
\]
\[
\text{20. continue}
\]
\[
\text{endif}
\]
For each segment, contact search is performed in etafind01, then the contact forces are applied in etacont01 or etacont02. The two subroutines, etafind and etacont are discussed in the following sections.

4.1 The contact search routine

The subroutine etafind01 does the global and local contact search for both contact types. After obtaining the master segment data, a check is made whether to do a global search or not.

    if (gscounter.ne.0) goto 110

If a global search has to be done, the offset of the bounding box is calculated and the bounding box is created.

    area = sqrt((y12*z14-z12*y14)**2 + (z12*x14-x12*z14)**2 + 
               (x12*y14-y12*x14)**2)
    offset=0.05*sqrt(area)*(0.05*GSC+0.95) + thick
    xmax = max(x1,x2,x3,x4) + offset
    xmin = min(x1,x2,x3,x4) - offset
    ...

After that, a loop is made over all the slave nodes to determine which slave node is within this box. If a slave node is already in contact with another master segment (stored in the contactflag), the slave node is not taken into account and no check is made.

    do 100 islave=1,nslave
        if (contactflag(slave(islave)).eq.1) goto 100
        xs=Xold(1,slave(islave))
        if ((xs.gt.xmax).or.(xs.lt.xmin)) goto 100
        ...
        ipair=ipair+1
    100 continue

When the number of potential slave nodes exceeds the maximum number of contact pairs (MAXCPAIR is set to 1000), an error warning is issued and the calculation is terminated.

    if (ipair.gt.MAXCPAIR) then
        write (ioout,1000) mn1,mn2,mn3,mn4
        write (ioout,1001)
        istrat=-1
        call exit
    1000 continue
    1001 continue
endif

cslave(ipair)=slave(islave)

100 continue

gpair=ipair

After the global search is done or skipped, the local search is performed. If
the global search did not result in any possible slave nodes for the master
segment, the local search is not performed and the subroutine is ended.

110 if (gpair.le.0) return

If possible slave nodes are found, a loop is made over all the possible slave
nodes to calculate their distance to the master segment. This is done by
projecting the slave node on the master segment using a modified Newton
iteration [7]. The distance between the slave node and the defence node is
given by:

\[ dist = |x^d - x^s| = \sqrt{(x^d_i(\xi, \eta) - x^s_i)^2} \quad i = 1, 3 \]  \hspace{1cm} (4.1)

(summation over i). The coordinates of the defence node are determined by
calculating the minimum of the squared distance \(D\) between \(x^d\) and \(x^s\).

\[ D(\xi, \eta) = (x^d_i(\xi, \eta) - x^s_i)^2 \quad i = 1, 3 \]  \hspace{1cm} (4.2)

\[ \frac{\partial D}{\partial \xi} = 2 \frac{\partial x^d_i}{\partial \xi} (x^d_i - x^s_i) = 0 \]  \hspace{1cm} (4.3)

\[ \frac{\partial D}{\partial \eta} = 2 \frac{\partial x^d_i}{\partial \eta} (x^d_i - x^s_i) = 0 \]  \hspace{1cm} (4.4)

Because the shape functions for the 4-node contact segment are bilinear in \(\xi\)
and \(\eta\), the derivative of the shape functions needed for the Newton-Raphson
iteration can be calculated numerically. This is done as follows. Take \(\eta = -1\)
and compute \(\xi\) from Eq.(4.3).

\[ \frac{\partial x^d_i}{\partial \xi} (x^s_i - x^d_i)|_{\eta=-1} = 0 \quad \rightarrow \quad \xi|_{\eta=-1} = C_1 \]  \hspace{1cm} (4.5)

So that we can calculate

\[ F_{N_1} = \frac{\partial x^d_i}{\partial \eta} (x^s_i - x^d_i)|_{(-1, C_1)} \]  \hspace{1cm} (4.6)

Unless the projection is correct, we find that \(F_{N_1} \neq 0\). If we do the same for
\(\eta = 1\) we get \(\xi = C_2\) and another expression for \(F_N\):

\[ F_{N_2} = \frac{\partial x^d_i}{\partial \eta} (x^s_i - x^d_i)|_{(1, C_2)} \]  \hspace{1cm} (4.7)
Unless \( F_{N_2} = 0 \), in case we have found the projection, we now have created an iteration procedure to find the new estimate for \( \eta \).

\[
\eta_{i+1} = \eta_i - \frac{2\eta_i}{F_{N_2} - F_{N_1}}
\]  

This modified Newton iteration, where the derivative is calculated numerically is implemented in the subroutine:

```plaintext
eta = -1.
ex = -(C1+C3*eta+C5*eta*eta)/(C2+C4*eta+C6*eta*eta)
fn1 = C7+C3*xi+C9*eta+2*C5*xi*eta+C4*xi+xi+C6*xi*xi*eta
if (abs(fn1).le.tol1) goto 20
if (abs(fn1old).lt.abs(fn1)) goto 200
fn1old=fn1
eta = -eta
ex = -(C1+C3*eta+C5*eta*eta)/(C2+C4*eta+C6*eta*eta)
fn2 = C7+C3*xi+C9*eta+2*C5*xi*eta+C4*xi+xi+C6*xi*xi*eta
if (abs(fn2).le.tol1) goto 20
if (abs(fn2old).lt.abs(fn2)) goto 200
fn2old=fn2
eta = -eta-2*eta*((fn1)/(fn2-fn1))
goto 10
```

If the projection is on this segment, the unit outward normal is calculated by taking the cross product of the derivatives in \( \xi \) and \( \eta \) of the shape function and normalize it.

```plaintext
tanx1 = Ax+eta*Bx
tanet1 = Cx+xi*Bx
...
normx = tanx1*tanetz - tanx1*tanetay
...
length = sqrt(normx+normy+normz+normz+normz)
nx = normx/length
...```

Now the distance to the master segment is calculated. If the distance is less than or equal to zero, the slave node is in contact with this segment. To prevent slave nodes at the other (negative) side of the master segment to be seen as contacting nodes and then to be pushed in the wrong direction, a maximum penetration is allowed. Compared to the maximum penetration
with the penalty method, this maximum can be very small as tests showed only very small penetrations (approximately $10^{-6}$).

$$\text{dist} = (xs-\text{xd})*n1x+(ys-\text{yd})*n1y+(zs-\text{zd})*n1z - \text{thick}$$

if (thick.gt.0) then
    MAXPENETR = -0.5*thick
else
    MAXPENETR = -0.1
endif

The contact flag for this slave node is set to 1, in order to prevent unnecessary calculations. The local coordinates $\xi$ and $\eta$ as well as the unit outward normal are stored to calculate the contact force.

if ((dist.le.0).and.(dist.gt.MAXPENETR)) then
    npair=npair+1
    inode=invslave(cslave(islave))
    contactflag(inode)=1
    contacting(npair)=cslave(islave)
    defence(1,npair)=xi
    defence(2,npair)=eta
    norm(1,npair)=n1x
    norm(2,npair)=n1y
    norm(3,npair)=n1z
endif

The contact data of the contact pairs is now passed on to the contact force routine.

4.2 The contact force routine

After finding which slave nodes come in contact with the master segment, the contact force is calculated in the contact force routines etacont01 or etacont02, depending on the contact type. The two subroutines are basically the same, but type 2 assumes the displacements of the master segment to be prescribed. It can be seen as a rigid foundation, thereby eliminating much of the calculations. Description of the contact force routine will therefore be done based on the routine for type 1 only.

By entering the subroutine, the master segment is checked for contacts. If no slave nodes are in contact with the segment the routine is ended. Also when the time step is zero (in the first cycle), no contact forces can be calculated.
After obtaining the contact data for the master segment, a loop is made over all the contact pairs. The coordinates for the slave node and the defence node are obtained and the penetration of the slave node is calculated.

\[
do 100 \text{ ipair}=1, \text{npair} \\
\text{islave}=\text{contacting(ipair)} \\
\text{xs}=\text{Xold}(1, \text{islave}) \\
\ldots \\
\text{xd} = \phi_1 x_1 + \phi_2 x_2 + \phi_3 x_3 + \phi_4 x_4 \\
\ldots \\
\text{penetr} = -((\text{xs}-\text{xd})*\text{norm}(1, \text{ipair})+(\text{ys}-\text{yd})*\text{norm}(2, \text{ipair})+ \\
* (\text{zs}-\text{zd})*\text{norm}(3, \text{ipair})-\text{thick})
\]

The mass of the defence node, the internal force on the defence node, the velocity of the defence node and the relative velocity between the slave node and the defence node are calculated.

\[
\text{massd} = (\text{mass}(1, \text{mn1})*\phi_1+\text{mass}(1, \text{mn2})* \\
* \phi_2+\text{mass}(1, \text{mn3})*\phi_3 + \\
* \phi_4)/\text{phibar2} \\
do 10 \text{ i}=1,3 \\
\text{ford}(i)=\text{massd}*((\phi_1/\text{mass}(1, \text{mn1}))* \\
* (\text{fore}(i, \text{mn1})-\text{fori}(i, \text{mn1}))+ \\
\ldots \\
\text{Veld}(i) = \phi_1*\text{Vold}(i, \text{mn1}) + \phi_2*\text{Vold}(i, \text{mn2}) + \\
* \phi_3*\text{Vold}(i, \text{mn3}) + \phi_4*\text{Vold}(i, \text{mn4}) \\
\text{dVel}(i) = \text{Veld}(i) - \text{Vold}(i, \text{islave}) \\
10 \text{ continue}
\]

The forces on the contact pair as well as the relative velocity are projected along the unit outward normal.

\[
\text{fordn} = \text{ford}(1)*\text{norm}(1, \text{ipair}) + \ldots \\
\text{forsn} = (\text{fore}(1, \text{islave})-\text{fori}(1, \text{islave}))*\text{norm}(1, \text{ipair}) + \\
\ldots \\
\text{dveln} = \text{dVel}(1)*\text{norm}(1, \text{ipair}) + \ldots
\]

Now we can calculate the normal contact force to impose the non-penetration condition.

\[
\text{forces} = ((\text{masss}*\text{massd})/(\text{masss}+\text{massd}))* (\text{fordn}/\text{massd} - \\
* \text{forsn}/\text{masss} + \text{dveln}/\text{dtnew}+\text{penetr}/(\text{dtnew}^2))
\]

20
When damping is included, the damping force is added to the normal contact force and restricted to be non-positive before being written into its components and put on the slave node.

\[
\text{if (damping.ne.0) then}
\]
\[
\text{forcs = forcs + damping*dveln}
\]
\[
\text{endif}
\]
\[
\text{if (forcs.lt.0.) forcs=0.}
\]
\[
\text{forc(i,islave) = forcs*norm(1,ipair)}
\]

When friction is included the tangential velocity and forces are calculated. To calculate the tangential component of a vector \( \mathbf{a} \) when the vector in global coordinates is known as well as the normal component, the tangential vector can be calculated by:

\[
\mathbf{a} = \mathbf{a}_n + \mathbf{a}_t \quad \rightarrow \quad \mathbf{a}_t = \mathbf{a} - \mathbf{a}_t \mathbf{N}_1
\]  

(4.9)

where the subscript \( t \) denotes the tangential component and the subscript \( n \) denotes the normal component.

\[
\text{if (sfric.ne.0.) then}
\]
\[
\text{do 20 i=1,3}
\]
\[
\text{dVtan(i) = dVel(i) - dveln*norm(i,ipair)}
\]
\[
\text{Fstan(i) = (fore(i,islave)-fori(i,islave)) -}
\]
\[
\text{* forsn*norm(i,ipair)}
\]
\[
\text{Fdtan(i) = ford(i) - fordn*norm(i,ipair)}
\]
\[
\text{20 continue}
\]

The force necessary to enforce the sticking condition is calculated, as well as the maximum static or dynamic friction force, depending on whether the contact is sticking or already sliding. If the necessary force is larger then the maximum static force, slip will occur and the friction force is limited to the dynamic friction force.

\[
dvelt = \sqrt{dVtan(1)*dVtan(1) + \ldots}
\]
\[
forst = \sqrt{Fstan(1)*Fstan(1) + \ldots}
\]
\[
fordt = \sqrt{Fdtan(1)*Fdtan(1) + \ldots}
\]
\[
Ffric = ((masss*massd)/(masss+massd))*( fordtn/massd -
\]
\[
* forst/masss + dvelt/dtnew )
\]
\[
inode=invslave(islave)
\]
\[
\text{if (slipflag(inode).eq.0) then}
\]
\[
Ffricmax = sfric*forssn
\]
else
    Ffricmax = dfric*forsn
endif

if (Ffric.gt.Ffricmax) then
    Ffric = dfric*forsn
    slipflag(inode)=1
else
    slipflag(inode)=0
endif

In case of small relative sliding, the friction force is projected in the opposite
direction of the relative sliding direction and added to the total contact force
on the slave node.

    if (dvelt.ne.0) then
        tan(1) = dVtan(1)/dvelt
        ...
        forc(1,islave) = forc(1,islave) - Ffric*tan(1)
        ...
    endif

Now that the total contact force (including damping and friction if necessary)
is known, the total contact force on the defence node is distributed over the
master nodes.

    if (forcs.ne.0.) then
        forcd = -forcs
        Bigphi1 = (mass(1,mn1)*phi1)/(massd*phibar2)
        ...
        if ((sfric.ne.0).and.(dvelt.ne.0)) then
            forcedx = forcd*norm(1,ipair) + Ffric*tan(1)
            ...
        else
            forcedx = forcd*norm(1,ipair)
            ...
        endif
    endif
    forc(1,mn1) = forc(1,mn1) + Bigphi1*forcedx
    ...
endif

This last part is of course ommitted in the calculation of the contact force in
case of contact type 2.
Chapter 5

Test results

To test and evaluate the contact algorithm several different tests were performed. First of all the implemented theory has to be proved, for which some simple tests were performed. The first two tests show the results of a plate pressed on or impacting another plate. The third test shows a more difficult problem compared with an analytical solution and the results from ABAQUS, an implicit FE-program. It also shows the solution obtained with the penalty approach implemented in ETA. The fourth test shows the result of friction. Then some demos are shown to demonstrate the robustness of the algorithm under severe conditions. They also show some of the possible applications of the ETA code. The input files of the test problems (not of the demo problems) can be found in Appendix B.

Figure 5.1: Two contacting plates, top view

23
5.1 Normal contact force

To validate the calculation of the normal contact force, two simple tests were performed. They consist of two plates, one larger than the other, as shown in Fig. (5.1). The nodes of the small plate are the slave nodes and the larger plate is the master segment. All the degrees of freedom of the master are fixed, the slave nodes are free.

5.1.1 Plate pressed on master segment

The first test is a quasi-static contact problem, as the two plates are already in contact and an external force of magnitude 1 is slowly applied to all the four slave nodes. This force is plotted in Fig. (5.2) together with the reaction force on one of the master nodes. As can be seen, the two forces are exactly the same, as one would expect. Although this test is the simplest test one can perform, it is also one of the most important tests, as it shows the correctness of the calculation of the contact force due to quasi-static loading.

5.1.2 Plate falling on master segment

The second test is made of the same plates, but now the initial position of the slave segment is 1 above the master segment and is given an initial velocity of 1000. Thus at \( t = 0.001 \) the slave hits the master and is bounced back. Part of the energy is absorbed by the internal (elastic) energy of the plate element, part of it is returned in kinetic energy, so that the impacting plate will bounce back. The energy transfer depends on the stiffness of the plates, as is shown in Fig. (5.3).
Figure 5.3: Displacement of the falling plate

5.1.3 Beam on rigid wall

The two tests above demonstrate the validity of the contact algorithm for very simple problems. The next problem, for which an analytical solution is known, concerns a bar or beam, impacting a rigid wall as shown in Fig.(5.4). This test is also compared with the result from another FE-program and the results obtained using the penalty method. In ETA, the beam is modelled with shell elements, as this is the only element currently available. The geometry of the beam is:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>10.</td>
</tr>
<tr>
<td>Width</td>
<td>0.1</td>
</tr>
<tr>
<td>Height</td>
<td>1.0</td>
</tr>
<tr>
<td>Young's modulus</td>
<td>1.e6</td>
</tr>
<tr>
<td>Poisson ratio</td>
<td>0.0</td>
</tr>
<tr>
<td>Density</td>
<td>0.01</td>
</tr>
</tbody>
</table>

If we look at the numerical solution obtained by ETA compared to the analytical solution of a rigid bar, as shown in Fig.(5.5), we see a good comparison.

![Beam against rigid wall, model](image)

Figure 5.4: Beam against rigid wall, model
Figure 5.5: Beam against rigid wall, contact force with Lagrange method. $L_1 = \text{ETA, scalefactor} = 0.8$; $L_2 = \text{analytical solution}$.

except in the last part, where the contact force drops to zero and peaks before dropping back to zero again. This is due to vibrations in the beam caused by the impact of the beam to the wall. The severity of these vibrations can be damped numerically by reducing the time step of the integration. If we reduce the time step scalefactor by 10, the peak of the contact force is gone, as shown in Fig.(5.6).

This problem was also solved using the implicit FE code ABAQUS [3]. Now the beam is modelled using beam elements and the wall is made of a rigid wall element. First the calculation was performed without maximum integration step. ABAQUS integrated from $t = 0$ to $t = 0.0025$ in 21 steps, resulting in a very coarse approximation of the analytical solution, see Fig.(5.7). Limiting the maximum step size to $\Delta t = 0.00005$, forcing ABAQUS to use at least 50 steps to integrate over the interval, a much better result is obtained as shown in Fig.(5.7).

These figures show the good comparison between the calculations performed by ETA and the calculations performed by ABAQUS as long as the time step is small enough. It also shows the advantage of using an explicit code for dynamic analysis instead of an implicit code. Although the time step of the implicit code is unconditionally stable, the geometric nonlinearity forces the time step size to be small. Forcing ABAQUS to reduce the time step increases CPU-time. Using 21 or 54 steps results in approximately the same CPU-time as required by ETA (with scalefactor 0.08). Also because a great part of the time is used for I/O. Increasing the number of steps of
ABAQUS to 1000 (1/2 the number of cycles uses by ETA), while at the same time limiting the output to 50 plots, shows an approximately tenfold increase in CPU-time.

Finally the same calculation was performed in ETA, but now the contact force was calculated using a penalty method. To reduce severe vibrations the time step scalefactor was set to 0.08. Three runs were made, the first using a penalty parameter \( \kappa = 1.e6 \), the second using \( \kappa = 1.e5 \) and the third using \( \kappa = 1.e4 \). The choice of a penalty factor is in principle arbitrary. The larger the penalty factor the better the non-penetration condition is satisfied. However, a large penalty parameter causes severe vibrations and reduces the critical time step. These severe vibrations caused by a penalty parameter of \( \kappa = 1.e6 \) are shown in Fig.(5.8). Reducing this penalty parameter smoothen these vibrations, but a too small penalty parameter causes too large penetrations and a 'soft' contact, Fig.(5.9).

These tests show the major disadvantage of the penalty parameter. Summarizing: a too large penalty parameter causes severe vibrations and reduces the critical time step. However, a too small penalty parameter creates a too soft contact and causes large penetrations. The optimum penalty parameter depends on the problem, especially on the stiffness of the two elements in contact. Therefore one always has to investigate the influence of this parameter, especially in explicit FE codes.
Figure 5.6: Beam against rigid wall, contact force with Lagrange method. $L_1 = \text{ETA}, \text{ scalefactor} = 0.08; L_2 = \text{analytical solution}$.

Figure 5.7: Beam against rigid wall, contact force with Lagrange method. $L_1 = \text{ABAQUS using 21 steps}; L_2 = \text{ABAQUS using 54 steps}; L_3 = \text{analytical solution}$. 

28
Figure 5.8: Beam against rigid wall, contact force with penalty method in ETA. $L_1 =$ analytical solution; $L_2 =$ penalty parameter $\kappa = 1e6$.

Figure 5.9: Beam against rigid wall, contact force with penalty method in ETA. $L_1 =$ analytical solution; $L_2 =$ penalty parameter $\kappa = 1e5$; $L_3 =$ penalty parameter $\kappa = 1e4$. 
5.2 Friction

To evaluate the calculation of the friction force, a simple test of a plate sliding over other plates (a rigid foundation) is performed, Fig.(5.10). The sliding plate is pressed against the master segments with a constant normal force, while a tangential force on the slave nodes increases from zero to half the normal force in $t = 0.0$ to $t = 0.0005$. The force is then reduced back to zero at time $t = 0.001$. The static friction coefficient is set to 0.4, while the dynamic friction coefficient is 0.3. If we look at the velocity of the sliding plate, Fig.(5.11), we see a (nearly) sticking situation until the maximum static friction force is reached. Then sliding occurs with increasing velocity and acceleration, until the tangential force reduces. Finally a sticking state is reached again. Looking at the tangential velocity in more detail, we would see small sliding of the plate. This is due to the decoupling of the contact force, just like small penetrations can occur. Also this small relative sliding is necessary in order to determine the direction of the friction force.

![Graph of tangential velocity vs time](image)

Figure 5.11: Sliding of a plate, tangential velocity
5.3 Ball impacting on plate

The test problems described above are meant to prove the correct behaviour and validity of the implemented contact algorithm. To test the contact algorithm more severely and to demonstrate the possibilities some more complicated tests were performed. The tests can also be found at the demo directory at the Delft University of Technology, Faculty of Aerospace Engineering.

The first demonstration shows a ball falling on an elastic plate, as shown in Fig.(5.12). The plate is clamped at the left end and the ball is given an initial velocity downwards. The plate is defined as the master segment, the nodes at the bottom side of the ball are defined as the slave nodes. The ball falls on the plate and pushes it down, Fig.(5.13). After maximum plate deflection, the plate pushes the ball up until release of the ball. Although relatively simple, it shows contact and release of contact.

A modified version of this problem is to place a second plate under the first plate. The upper plate, which is a master surface for the ball, is now also the slave surface for the second plate. Because of the ball impacting the upper plate, the first plate will come into contact with the second plate. This
is considered a difficult constraint for a contact algorithm. If the constitutive model for the two plates is elastic-plastic and if the yield stress of the plates is chosen low enough, severe plastic deformation will occur. Placing the ball on the side of the plates causes complex behaviour of the total contact-impact problem, Fig.(5.14).

5.4 A sample stamping problem

An example of future applications for ETA is the modeling of stamping. A simple stamping problem is shown in Fig.(5.15). A constant force acts on the blankholder to press the sheet to the die. The punch is moved downward with a constant prescribed velocity. When the punch comes in contact with the sheet, the sheet moves with the punch to be brought in its final deformed state, Fig.(5.16). Looking at the deformed state of Fig.(5.16) more closely we see the disadvantage of the use of a node-to-segment contact instead of a real segment-to-segment contact. As only the slave nodes (of the sheet) cannot penetrate the master segment, the elements of the sheet can penetrate due to the curved master segment. Increasing the number of elements of the slave surface will reduce these penetrations.

The stamping problem is one of the more difficult problems to be solved using the finite element method. It includes contact, sliding over curved surfaces and plasticity including elastic spring-back. Because of these problems the stamping problem is, together with crashing, one of the most difficult research topics in dynamic finite elements.
Figure 5.15: Stamping problem, model

Figure 5.16: Stamping problem, deformed configuration
5.5 Two tubes impacting

The last example shown in this report is the impact of two tubes perpendicular to each other. This can be seen as a sample crashing problem and shows some of the difficulties of this kind of analysis.

The first calculations are done with two tubes, made of 180 shell elements each as shown in Fig. (5.17). The two tubes have the properties given in Table (5.1). The tube on the left side is much stiffer than the tube on the right side. Also the right tube has a low yield stress, allowing for elastic-plastic deformations. The tubes have both an initial impact velocity of 8000. Because of the low stiffness and low yield stress of the right tube, this tube

<table>
<thead>
<tr>
<th></th>
<th>Left tube</th>
<th>Right tube</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length (mm)</td>
<td>40.</td>
<td>40.</td>
</tr>
<tr>
<td>Radius (mm)</td>
<td>10.</td>
<td>10.</td>
</tr>
<tr>
<td>Thickness (mm)</td>
<td>1.0</td>
<td>0.1</td>
</tr>
<tr>
<td>$E$ (MPa)</td>
<td>200000.</td>
<td>70000.</td>
</tr>
<tr>
<td>$\nu$</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>$\rho$ (kg/mm$^3$)</td>
<td>7.8e-9</td>
<td>3.65e-9</td>
</tr>
<tr>
<td>$\sigma_y$ (MPa)</td>
<td>-</td>
<td>175.0</td>
</tr>
</tbody>
</table>

Table 5.1: Properties tubes
Figure 5.18: Two impacting tubes, full collapse

collapses onto the stiffer tube. After the large deformations of the impacting side of the right tube, a lot of energy is absorbed and the two tubes move in the direction of the initial velocity of the left tube. Due to inertia, the back side of the right tube collapses as well. See Fig.(5.18).

In order to test both the contact algorithm as well as the element performance, the yield stress of the left tube is reduced to 105 MPa and the plasticity model is made elastic-perfectly plastic. Here the deformations become too large, causing instability of the element and termination of the calculation. Increasing the number of elements of the left tube to 1080 elements improves the stability, terminating the calculation at a more severe deformation. As can be seen in Fig.(5.19) the angle between certain elements is very sharp, causing the calculation to be terminated. Although this test puts severe demands on the contact algorithm it performs very well.
Figure 5.19: Two impacting tubes, maximum deformation due to element instability
Chapter 6

Conclusions and recommendations

Based on the test results showed in this report, the conclusion can be made that the implemented contact algorithm is correct and gives satisfactory results. Studying the beam impacting a rigid wall shows good results compared to the analytical solution as well as to the results obtained by ABAQUS. This test also demonstrates the difference between the solution obtained by an explicit and an implicit FE code and the difference between the use of a penalty method or a Lagrange method. It shows the vibrations in the beam due to the impact, not taken into account by the analytical solution.

The more complex problems show the good behaviour and the robustness of the implemented algorithm. Both the multiple contact system (the ball on the two plates) as well as contact due to large deformations (the two impacting cylinders) show a satisfactory behaviour as far as the contact-impact is concerned. Of course, no analytical solution is available and comparison with another FE code would be much influenced by other effects, like the shell element used and the integration of the equations in time.

Based on the tests shown in the previous chapter and with the experience obtained while using ETA the following recommendations can be made.

The only problem encountered during extensive testing of the contact algorithm was the maximum size of the data set. It turns out that if a contact interface is defined using of a lot of master segments and a large set of slave nodes, the maximum size of the data set in MEM-COM is reached long before the imposed maxima on the master segments and slave nodes. To solve this problem, it is recommended that the definition of the master segments is stored in a different data set. This can also create the possibility to display the master segments with the post-processing program BASPL.
Due to the implementation of computational restart, plasticity and contact, the ETA code is now reasonably well developed to handle some interesting problems, but lacks a lot of functionality due to the limitation in elements, as only one shell element is implemented. This limits the applicability of the program. Development of the ETA code should be aimed at the implementation of new elements like a volume element and a beam element. Also the implementation of rigid bodies is recommended to speed up certain type of calculations.

The implemented shell element is not completely free of problems, especially the so-called junctures may cause problems. Implementation of new shell elements is also recommended. Two types of elements are in favour: the new 6 DOF shell element developed by Gert Rebel and a shell element capable of large deformations.

The implemented plasticity, the Besseling fraction model, is one of the best models available for small deformations. This could turn out to be very useful for stamping problems and cyclic loading in the plastic region. However, it uses a lot of CPU time and uses a lot of storage space. Therefore implementation of a simple plasticity model (elastic-perfectly plastic or isotropic hardening) is recommended for problems like crashing. As implementation of this is relatively simple (one could modify the existing routines) this can be done as a preparing excercise.

Because the ETA code is based on the same input syntax and also based on the data base management system MEM-COM [5] as the implicit FE code B2000 [4, 6], it could be an interesting research topic to connect the two codes. This way, one might be able to combine (quasi-)static and dynamic analysis (e.g. static buckling, dynamic post-buckling).

Finally, the author wishes to express his hope that the ETA code will not be put to rest for another two years as the ETA code is considered to be potentially one of the best explicit finite element codes for research institutes available today. The access to the source code, the use of the database manager MEM-COM and the command driven input syntax make ETA a very user friendly program and preferred over 'black box' programs from commercial companies.
Bibliography


Appendix A

Source files

A.1 b2ipcontact.F

```fortran
subroutine b2ipcontact(ulist,iul,ibr,nnb,irad,istat)
  implicit none
  integer ulist,iul,ibr,nnb,irad,istat

  #include "b2limits.ins"
  #include "b2extern.ins"
  #include "b2adir.ins"
```

#include "dmm.ins"
#include "b2io.ins"
#include "b2csi.ins"
#include "b2etapar.ins"

character label*32, err*20
real*8 zero, flo, mthick, sthick, sfric, dfric, damp
integer i, j, k, l, m, n, rem, iseg, islave
integer idtmp1, idtmp1, idtmp2, idtmp3, idtmp3, idtmp3
integer nmemory, icstat, ist, int, i
integer istorem, istores
integer nmaster, nsalve, GSC, ctype
integer totsurface, totmaster, totslave
integer seg, nm, nm1, nm2, nm3, nm4
integer ns1, ns2, ns3, dirflag

data zero /0.0/
data err/'***IDL-ERROR (INIT) '/

c 1000 format(///"Contact definition, branch= ',i5/64('*,')")
1020 format(///"Definition of master segements, surface = ',i3)
1021 format(/' Node_1 Node_2 Node_3 Node_4'
1022 format(1x,4(i8,1x))
1030 format(//"Definition of slave nodes, surface = ',i3/
1031 format(//' Node_from Node_to Node_step'
1032 format(1x,3(i8,1x))
1040 format(//'Contact surface data:'
1041 format(//' Contact type ',i3)
1042 format(//' Contact thicknesses : ',2(1pe12.4,2x))
1043 format(//' Friction coefficients: ',2(1pe12.4,2x))
1044 format(//' Damping coefficient : ',1pe12.4,2x)
1050 format(//'Master segments and slave nodes for branch',i3,' saved.\')
c
istat=0
if (iist.gt.0) write(ioout,1000) ibr
if (mnb.1.e.0) then
  write (ioerr,*)'***IDL-FATAL No nodes defined, cannot continue'
  istat=-1
  call exit
endif
istorem=0
istores=0
CSID=0
totmaster=0
totslave=0
totsurface=0
nmaster=0
nslave=0
cstype=0
dirflag=1
c
c set default values
c
gsc=21
mthick=zero
sthick=zero
damp=zero
dfric=zero
sfric=zero
c
c reserve memory in DMM
c

mmemory=4*MAXCMASTER
call dmmall(nmemory, idtmp1, ist)
iptmp1=dmmptr(idtmp1)
call b2setint(0, irad(iptmp1), nmemory)
mmemory=MAXCSLAVE
call dmmall(nmemory, idtmp2, ist)
iptmp2=dmmptr(idtmp2)
call b2setint(0, irad(iptmp1), nmemory)
mmemory=3
goto 10
8 write (ioerr,* ) err, 'Syntax error at key ',key
9 istat=-1
10 icstat=f cigetkey(key,lkey)
   if (icstat.lt.0) goto 8
c
c read contact surface number and contact type
c
15 if (key(1:4).eq.'CSID') then
   icstat=f cigetint(int,1)
   if (icstat.lt.0) goto 8
   CSID=CSID+1
   if (int.ne.CSID) goto 8
   if (CSID.gt.MAXCSURFACE) then
      write (ioerr,* ) err,'Contact segment number exceeds maximum number of contact segments ',CSID
      goto 9
   endif
   goto 10

   else if (key(1:4).eq.'CTYP') then
   icstat=f cigetint(int,1)
   if(icstat.lt.0) goto 8
   cstype=int
c
c check if contact type exists
  if ((ctype.eq.1).and.(ctype.ne.2)) then
    write (ioerr,*) err,'Contact type ','int,' not defined.'
    istat=-1
    call exit
  endif
  goto 10

c else if (key(1:4).eq.'THIC') then
  icstat=fcsigetfloat(flo,1)
  if(icstat.lt.0) goto 8
  mthick=flo
  goto 10

c else if (key(1:4).eq.'MTHI') then
  icstat=fcsigetfloat(flo,1)
  if(icstat.lt.0) goto 8
  mthick=flo
  goto 10

c else if (key(1:4).eq.'STHI') then
  icstat=fcsigetfloat(flo,1)
  if(icstat.lt.0) goto 8
  stthick=flo
  goto 10

c else if (key(1:4).eq.'FRIC') then
  icstat=fcsigetfloat(flo,1)
  if(icstat.lt.0) goto 8
  dfri=flo
  sfri=flo
  goto 10

c else if (key(1:4).eq.'DFRI') then
  icstat=fcsigetfloat(flo,1)
  if(icstat.lt.0) goto 8
  dfri=flo
  goto 10

c else if (key(1:4).eq.'SFRI') then
  icstat=fcsigetfloat(flo,1)
  if(icstat.lt.0) goto 8
  sfri=flo
  goto 10

c else if (key(1:4).eq.'DAMP') then
  icstat=fcsigetfloat(flo,1)
  if(icstat.lt.0) goto 8
damp=flo
goto 10

else if (key(1:4).eq.'GSEA') then
  icstat=fcigetint(int,1)
  if(icstat.lt.0) goto 8
  gsc=int
  goto 10

else if (key(1:3).eq.'DIR') then
  icstat=fcigetint(int,1)
  if(icstat.lt.0) goto 8
  if (int.eq.-1) then
    dirflag=-1
  else
    dirflag=1
  endif
  goto 10

else if (key(1:4).eq.'MAST') then
  iseg=0
  write (ioout,1020) CSID
  write (ioout,1021) iseg
  icstat=fcigetkey(key,1key)
  if (icstat.lt.0) goto 8
  if (icstat.ne.NUMBEREAD) then
    nmaster=iseg
    dirflag=1
    goto 15
  endif
  iseg=iseg+1
  if (dirflag.eq.-1) then
    icstat=fcigetint(nm4,1)
    irad(iptmpl+(3+(iseg-1)*4))=nm4
    icstat=fcigetint(nm3,1)
    if (icstat.lt.0) goto 8
    irad(iptmpl+(2+(iseg-1)*4))=nm3
    icstat=fcigetint(nm2,1)
    if (icstat.lt.0) goto 8
    irad(iptmpl+(1+(iseg-1)*4))=nm2
    icstat=fcigetint(nm1,1)
    if (icstat.lt.0) goto 8
    irad(iptmpl+(0+(iseg-1)*4))=nm1
  else
    icstat=fcigetint(nm1,1)
    irad(iptmpl+(0+(iseg-1)*4))=nm1
    icstat=fcigetint(nm2,1)
    if (icstat.lt.0) goto 8
    irad(iptmpl+(1+(iseg-1)*4))=nm2
icstat=fcigetint(nm3,1)
if (icstat.lt.0) goto 8
irad(iptmp1+(2+(iseg-1)*4))=nm3
icstat=fcigetint(nm4,1)
if (icstat.lt.0) goto 8
irad(iptmp1+(3+(iseg-1)*4))=nm4
endif

check if segment no triangle
if ((nm1.eq.nm2).or.(nm1.eq.nm3).or.(nm1.eq.nm4).or.
    (nm2.eq.nm3).or.(nm2.eq.nm4).or.(nm3.eq.nm4)) then
  write(ioerr,*) err,'Double declaration of master nodes at
  segment ',iseg
  istat=-1
  call exit
endif
if ((nm1.gt.nnb).or.
    (nm2.gt.nnb).or.
    (nm3.gt.nnb).or.
    (nm4.gt.nnb)) then
  write(ioerr,*) err,'Wrong declaration of master segment
  at segment ',iseg
  istat=-1
  call exit
endif
istorem=1
write (ioout,1022) nm1,nm2,nm3,nm4
goto 30

read slave nodes
else if (key(1:4).eq.'SLAV') then
  islave=0
  write (ioout,1030) CSID
icstat=fcigetkey(key,1key)
if (icstat.lt.0) goto 8
if (icstat.ne.NUMBEREAD) then
  nslave=islave
goto 15
endif
icstat=fcigetint3(key,1key,ns1,ns2,ns3)
if (icstat.lt.0) goto 8
d0 45 i=ns1,ns2,ns3
  islave=islave+1
  irad(iptmp2+(islave-1))=i
  continue
if ((ns1.gt.nnb).or.(ns2.gt.nnb)) then
  write(ioerr,*) err,'Wrong declaration of slave nodes
  at segment ',iseg
* at contact surface 'CSID

    istat=-1
    call exit
endif
istores=1
write (ioout,1031)
write (ioout,1032) ns1,ns2,ns3
goto 40

else if(key(1:4).eq.'ENDC') then
    if (ctype.eq.0) then
        write (ioerr,*) err,'Contact type not defined for surface',
        'CSID
        istat=-1
        call exit
endif
    if (nnmaster.eq.0) then
        write (ioerr,*) err,'Master surface not defined for surface',
        'CSID
        istat=-1
        call exit
endif
    if (nsslave.eq.0) then
        write (ioerr,*) err,'Slave nodes not defined for surface',
        'CSID
        istat=-1
        call exit
endif

store contact data in database

totsurface=totsurface+1
totmaster=totmaster+nnmaster
totslave=totslave+nsslave
call putbwi('CTYPE',ctype,ist)
call putbwi('GSC',gsc,ist)
call putbwi('NMASTER',nnmaster,ist)
call putbai('MASTER',nnmaster,irad(imp1),ist)
call putbwi('NSLAVE',nsslave,ist)
call putbai('SLAVE',nsslave,irad(imp2),ist)
call putbwf('MTHICK',mthick,ist)
call putbwf('STHICK',sthick,ist)
call putbwf('SFRIC',sfri,ist)
call putbwf('DFRIC',dfri,ist)
call putbwf('DAMPING',damp,ist)
label='CONT.1.'/i2lnc(CSID)
call puttab(iu1,label,0,ist)
call clrtb
write (ioout,1040)
write (ioout,1041) ctype
write (ioout,1042) mthick, sthick
write (ioout,1043) sfrie, dfrie
write (ioout,1044) damp

restore default values

gsc=0
mthick=zero
sthick=zero
sfrie=zero
dfrie=zero
damp=zero

goto 10

else if (key(1:3).eq.'END') then
  label='CDIR.'//i2lc(ibr)
call restab(iu1,label,0,ist)
call clrtb
call putbwi('TOTSURF',totsurface,ist)
call putbwi('TUTMASTER',totmaster,ist)
call putbwi('TOTSMASTER',totslave,ist)
call puttab(iu1,label,0,ist)
write (ioout, 1050) ibr
call dmmrid(idtmp1,ist)
call dmmrid(idtmp2,ist)
return
else
  write (ioerr,*) err,'Illegal CONTACT command: ',key(1:1key)
goto 9
endif

end
A.2 etacontact.F

subroutine etacontact(Xold,cdata,
* csurface,master,
* slave,forc,
* fori,Vold,
* mass,fore,
* norm,defence,
* contacting,cslave,
* gpair,contactflag,
* invslave,slipflag)

C2345678901234567890123456789012345678901234567890123456789012**
C 1 2 3 4 5 6 7
C
C Determines contacting nodes and calculates contact forces
C*******************************************************************************
C
c csurface(1,isurf) = contact type
c csurface(2,isurf) = number of master segments
c csurface(3,isurf) = number of slave nodes
c csurface(4,isurf) = global contact search per GCS cycles
c csurface(5,isurf) = gcs counter
C
c cdata(1,isurf) = thickness master segment
c cdata(2,isurf) = thickness slave segment
c cdata(3,isurf) = static friction coefficient
c cdata(4,isurf) = dynamic friction coefficient
c cdata(5,isurf) = contact damping factor
C
c master(4*TOTMASTER) = nodes of the master segments of all surfaces
C slave(TOTSLAVE) = global node numbers of slave nodes of all surfaces
C gpair(TOTMASTER) = number of contact pairs per master segment
C cslave(MAXPAIR,
C TOTMASTER) = (possible) contacting slave nodes
C invslave(NNODES,
C TOTSURFACE) = slave node number of global node number
C
C MODIFICATIONS
C+---------------------------------------------------------------------------+
C | Date  | Name | Comments |
C+---------------------------------------------------------------------------+
C | 1996  | PTG Volgers | creation, contact type 1 and 2 |
C+---------------------------------------------------------------------------+
C
C implicit none

49
#include "b2constants.ins"
#include "b2eta.ins"
#include "b2io.ins"
#include "b2etapar.ins"

real*8 Xold(NDOF,NNODES),Vold(NDOF,NNODES)
real*8 fori(NDOF,NNODES),fore(NDOF,NNODES),forc(3,NNODES)
real*8 cdata(5,TOTSURFACE), zero, mass
integer csurface(5,TOTSURFACE),master(4*TUTMASTER),
* slave(TUTSLAVE), cslave(MAXPAIR,TUTMASTER),
* gpair(TUTMASTER), slipflag(TUTSLAVE),
* invslave(NNODES,TOTSURFACE)
integer ctyp, imaster, isurf, GSC, i, contactflag(MAXSLAVE)

real*8 defence, norm
integer contacting, nslaveprev, mmasterprev
integer nmemory

1000 format(/**** WARNING, INITIAL CONTACT AT CONTACT SURFACE */,i4)
if (TOTSURFACE.eq.0) return
data zero /0.0/

nmemory=3*NNODES
call b2setfloat(zero,forc,nmemory)
nslaveprev=0
mmasterprev=0

Loop over all the contact surfaces
**********************************************************

do 100 isurf=1,TOTSURFACE
ctyp=csurface(1,isurf)
mmaster=csurface(2,isurf)
nslave=csurface(3,isurf)
call b2setint(0,contactflag,MAXSLAVE)

contact type 1: deformable master segments

if (ctyp.eq.1) then
  do 10 imaster=1,mmaster
    call etafind01(Xold ,csurface(4,isurf),
    master(1+4*(imaster-1+mmasterprev)),
    slave(1+nslaveprev),
    mass ,norm ,
    defence ,contacting ,
    cslave(1,imaster+mmasterprev),
    gpair(imaster+mmasterprev),
    contactflag ,cdata(1,isurf),
    invslave(1,isurf))

1000 continue

50
call etacont01(Xold, forc, 
* fori, Vold, 
* master(1+4*(imaster-1+nmasterprev)), 
* slave(1+nslaveprev), 
* mass, norm, 
* defence, contacting, 
* fore, cdata(1, isurf), 
* slipflag, invslave(1, isurf))
10 continue
endf

c contact type 2: master surface rigid

c if (ctyp.eq.2) then
do 20 imaster=1,nmaster

call etacont01(Xold, csurface(4, isurf), 
* master(1+4*(imaster-1+nmasterprev)), 
* slave(1+nslaveprev), 
* mass, norm, 
* defence, contacting, 
* cslave(1, imaster+nmasterprev), 
* gpair(imaster+nmasterprev), 
* contactflag, cdata(1, isurf), 
* invslave(1, isurf))

call etacont02(Xold, forc, 
* fori, Vold, 
* master(1+4*(imaster-1+nmasterprev)), 
* slave(1+nslaveprev), 
* mass, norm, 
* defence, contacting, 
* fore, cdata(1, isurf), 
* slipflag, invslave(1, isurf))
20 continue
endf

c contact type 3: penalty method, deformable master

c if (ctyp.eq.3) then
do 30 imaster=1,nmaster

call etacont01(Xold, csurface(4, isurf), 
* master(1+4*(imaster-1+nmasterprev)), 
* slave(1+nslaveprev), 
* mass, norm, 
* defence, contacting, 
* cslave(1, imaster+nmasterprev), 
* gpair(imaster+nmasterprev), 
* contactflag, cdata(1, isurf), 
* invslave(1, isurf))
call etacont03(Xold, forci, Vold, master(i+4*(imaster-1+nmasterprev)), slave(i+nslaveprev), mass, norm, defence, contacting, fore, cdata(i,isurf), slipflag, invslave(i,isurf))

continue

endif

c

nslaveprev = nslaveprev + nslave
nmasterprev = nmasterprev + nmaster
if ((time.eq.0).and.(npair.gt.0)) then
    write(ioout,1000) isurf
endif

if (csurface(5,isurf).gt.csurface(4,isurf)) csurface(5,isurf)=0

continue

c

return

end
A.3 etafind01.F

subroutine etafind01(Xold ,csurface ,
  master ,slave ,
  mass ,norm ,
  defence ,contacting ,
  cslave ,
  gpair ,
  contactflag ,cdata ,
  invslave )

Determine contacting nodes; global and local search

******************************

MODIFICATIONS

<table>
<thead>
<tr>
<th>Date</th>
<th>Name</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1996</td>
<td>PTG Volgers</td>
<td>creation, contact type 1 and 2</td>
</tr>
</tbody>
</table>

implicit none

#include "b2constants.ins"
#include "b2eta.ins"
#include "b2io.ins"
#include "b2etapar.ins"

real*8 Xold(NDOF,NNODES), mass, cdata(2)
integer master(4), slave(nslave), cslave(nslave),
  invslave(NNODES)
integer istat, islave, ipair, gpair, inode
integer csurface(2), GSC, gscounter, contactflag(nslave)

integer mn1,mn2,mn3,mn4
real*8 x1,x2,x3,x4,y1,y2,y3,y4,z1,z2,z3,z4,xs,ys,zs
real*8 x12,y12,z12,x14,y14,z14,area,offset
real*8 xmax,xmin,ymax,ymin,zmax,zmin
real*8 Ax,Ay,Az,Bx,By,Bz,Cx,Cy,Cz,Dx,Dy,Dz
real*8 C1,C2,C3,C4,C5,C6,C7,C9
real*8 eta,xi,tol1,masterthick,slavethick,thick
real*8 fn1,fn2,fn1old,fn2old,MAXPENETR

real*8 phi1,phi2,phi3,phi4,xd,yd,zd
real*8 tanxix,tanxiy,tanxiz,tanetax,tanetay,tanetaz
real*8 nlx,nly,nlz
real*8 normx,nomy,normz,length,dist

integer contacting(nslave)
real*8 defence(2,nslave), norm(3,nslave)

data tol1 /0.001/

c GSC=csurface(1)
gscounter=csurface(2)
masterthick=cdata(1)
slavethick=cdata(2)
thick=masterthick+slavethick
ipair=0
npair=0

1000 format(/** WARNING, maximum number of possible contacts
* at ',i4,i4,i4,i4,i4)
1001 format('/Reduce Global Search Counter or segment size.')

mn1=master(1)
mn2=master(2)
mn3=master(3)
mn4=master(4)

x1=Xold(1,mn1)
y1=Yold(2,mn1)
z1=Zold(3,mn1)
x2=Xold(1,mn2)
y2=Yold(2,mn2)
z2=Zold(3,mn2)
x3=Xold(1,mn3)
y3=Yold(2,mn3)
z3=Zold(3,mn3)
x4=Xold(1,mn4)
y4=Yold(2,mn4)
z4=Zold(3,mn4)

if (gscounter.ne.0) goto 110

c global contact search

c x12=x2-x1
y12=y2-y1
z12=z2-z1
x14=x4-x1
y14=y4-y1
z14=z4-z1

area = sqrt((y12*z14-x12*y14)**2 + (z12*x14-x12*z14)**2 +
* (x12*y14-y12*x14)**2)
offset = 0.05*sqrt(area)*(0.05*GSC+0.95) + thick
xmax = max(x1,x2,x3,x4) + offset
xmin = min(x1,x2,x3,x4) - offset
ymax = max(y1,y2,y3,y4) + offset
ymin = min(y1,y2,y3,y4) - offset
zmax = max(z1,z2,z3,z4) + offset
zmin = min(z1,z2,z3,z4) - offset

do 100 islave=1,nslave

c if slavenode already in contact, return

c if (contactflag(islave).eq.1) goto 100

c xs=Xold(1,slave(islave))
if ((xs.gt.xmax).or.(xs.lt.xmin)) goto 100
ys=Yold(2,slave(islave))
if ((ys.gt.ymax).or.(ys.lt.ymin)) goto 100
zs=Zold(3,slave(islave))
if ((zs.gt.zmax).or.(zs.lt.zmin)) goto 100
ipair=ipair+1
if (ipair.gt.MAXPAIR) then
  write (ioout,1000) mn1,mn2,mn3,mn4
  write (ioout,1001)
  istrate=-1
  call exit
endif
cslave(ipair)=slave(islave)
100 continue

c gspair=ipair

c local contact search

c constants

c 110 if (gspair.le.0) return

c Ax = -0.25*(x1-x2-x3+x4)
Ay = -0.25*(y1-y2-y3+y4)
Az = -0.25*(z1-z2-z3+z4)
Bx = 0.25*(x1-x2+x3-x4)
By = 0.25*(y1-y2+y3-y4)
Bz = 0.25*(z1-z2+z3-z4)
Cx = -0.25*(x1+x2-x3-x4)
Cy = -0.25*(y1+y2-y3-y4)
Cz = -0.25*(z1+z2-z3-z4)
Dx = 0.25*(x1+x2+x3+x4)
Dy = 0.25*(y1+y2+y3+y4)
$Dz = 0.25(z1+z2+z3+z4)$

$C2 = Ax*Ax+Ay*Ay+Az*Az$
$C4 = Ax*Bx+Ay*By+Az*Bz$
$C5 = Bx*Cx+By*Cy+Bz*Cz$
$C6 = Bx*Bx+By*By+Bz*Bz$
$C9 = Cx*Cx+Cy*Cy+Cz*Cz$

determine distance of slave nodes to master

do 200 islave=1,gspair
    xs=Xold(1,cslave(islave))
    ys=Xold(2,cslave(islave))
    zs=Xold(3,cslave(islave))

constants

$C1 = Ax*(Dx-xs)+Ay*(Dy-ys)+Az*(Dz-zs)$
$C3 = Ax*Cx+Bx*(Dx-xs)+Ay*Cy+By*(Dy-ys)+Az*Cz+Bz*(Dz-zs)$
$C7 = Cx*(Dx-xs)+Cy*(Dy-ys)+Cz*(Dz-zs)$

projection of slave node on master segment

fn1old=1000000.
fn2old=1000000.
eta = -1.
10   xi = -(C1+C3*eta+C5*eta*eta)/(C2+2*C4*eta+C6*eta*eta)
    fn1 = C7+C3*xi+C9*eta+2*C5*xi*eta+C4*xi*xi+C6*xi*xi*eta
    if (abs(fn1).le.tol1) goto 20

check for divergence

if (abs(fn1old).lt.abs(fn1)) goto 20
fn1old=fn1
eta = -eta
xi = -(C1+C3*eta+C5*eta*eta)/(C2+2*C4*eta+C6*eta*eta)
fn2 = C7+C3*xi+C9*eta+2*C5*xi*eta+C4*xi*xi+C6*xi*xi*eta
if (abs(fn2).le.tol1) goto 20
if (abs(fn2old).lt.abs(fn2)) goto 200
fn2old=fn2
eta = -eta-2*eta*((fn1)/(fn2-fn1))
goto 10

check projection on element and calculate defence node coordinates

slight offset to fill-in the gaps (1%)

20 if (thick.eq.0) then
    offset=0.
else
    offset=0.01
endif
if ((abs(eta).gt.1+offset).or.(abs(xi).gt.1+offset)) goto 200
phi1 = 0.25*(1-xi)*(1-eta)
phi2 = 0.25*(1+xi)*(1-eta)
phi3 = 0.25*(1+xi)*(1+eta)
phi4 = 0.25*(1-xi)*(1+eta)
xd = phi1*xi+phi2*x2+phi3*x3+phi4*x4
yd = phi1*y1+phi2*y2+phi3*y3+phi4*y4
zd = phi1*z1+phi2*z2+phi3*z3+phi4*z4

calculate set of unit vectors

tanxi = Ax+eta*Ex
tanxy = Ay+eta*Ey
tanxiz = Az+eta*Ex

tanetax = Cx+xi*Ex
tanetay = Cy+xi*Ey

tanetaz = Cz+xi*Ex

c norm = tanxi x taneta
	normx = tanxiy*tanetaz - tanxiz*tanetay
normy = tanxiz*tanetax - tanxix*tanetaz
normz = tanxix*tanetay - tanxiy*tanetax

c normalize unit outward vector n1 and tangent vector n2

c length = sqrt(normx*normx+normy*normy+normz*normz)
nix = normx/length
niy = normy/length
niy = normz/length

c calculate distance of slave node to master segment

c dist = (xs-xd)*n1x+(ys-yd)*n1y+(zs-zd)*n1z - thick
if (thick.gt.0) then
    MAXPENETR = -0.5*thick
else
    MAXPENETR = -0.1
endif
if ((dist.le.0).and.(dist.gt.MAXPENETR)) then

c slave node in contact with master segment

c store slave node, set of unit vectors and defence node coordinates

cmpair=mpair+1
contactflag(cslave(islave))=1
inode=invslave(cslave(islave))
contactflag(inode)=1
contacting(npair)=cslave(islave)
defence(1,npair)=xi
defence(2,npair)=eta
norm(1,npair)=n1x
norm(2,npair)=n1y
norm(3,npair)=n1z
endif

200 continue

c

return
end
A.4  etacont01.F

```fortran
subroutine etacont01(Xold , forc ,
   * fori , Vold ,
   * master , slave ,
   * mass , norm ,
   * defence , contacting ,
   * fore , cdata ,
   * slipflag , invslave )

C   Calculates contact forces for a segment
C   *************************************************************************
C
C   MODIFICATIONS
C   +---------------------------------------------------------------------+
C   |       Date | Name          | Comments                     |
C   +---------------------------------------------------------------------+
C   | 1996      | PTG Volgers   | creation, contact type 1     |
C   +---------------------------------------------------------------------+
C
C   implicit none
C
C   #include "b2constants.ins"
C   #include "b2eta.ins"
C
real*8 Xold(NDOF,NNODES), Vold(NDOF,NNODES)
real*8 fori(NDOF,NNODES), forc(3,NNODES), fore(NDOF,NNODES)
real*8 mass(NDOF,NNODES), norm(3,npair), defence(2,npair),
*       cdata(5)
integer master(4), slave(mslave), contacting(npair),
*       slipflag(mslave), invslave(NNODES)
C
integer nni,mm2,mm3,mm4, ipair, islave, inode
real*8 x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,z4,ys,zs,xd,yd,zd
real*8 eta,xi, phibar2,phi1,phi2,phi3,phi4
real*8 Bigphi1,Bigphi2,Bigphi3,Bigphi4
real*8 masses,massd,forfn,forst,vels,penetr,forces,forcd,
*       forcedx,forcedy,forcedz
C
real*8 ford(3),fordn,fordt,Veld(3),dVel(3),dVtan(3),dveln,dvelt ,
*       sfric,ffric,tan(3),Fdtan(3),Fstan(3),Ffric,Ffricmax,thick ,
*       damping
integer i
C
if (npair.eq.0) return
```

59
if (dtnew.eq.0) return

thick = cdata(1) + cdata(2)
sfric = cdata(3)
dfric = cdata(4)
damping = cdata(5)

nn1=master(1)
nn2=master(2)
nn3=master(3)
nn4=master(4)

xin=Xold(1,nn1)
yin=Xold(2,nn1)
zin=Xold(3,nn1)
x2=Xold(1,nn2)
y2=Xold(2,nn2)
z2=Xold(3,nn2)
x3=Xold(1,nn3)
y3=Xold(2,nn3)
z3=Xold(3,nn3)
x4=Xold(1,nn4)
y4=Xold(2,nn4)
z4=Xold(3,nn4)

do 100 ipair=1,npair
   isslave=contacting(ipair)
   xs=Xold(1,islave)
   ys=Xold(2,islave)
   zs=Xold(3,islave)
   xi = defence(1,ipair)
   eta= defence(2,ipair)
   phivar2 = 0.25*(1+eta*eta+xi+xi+xi+xi+eta*eta)
   phi1 = 0.25*(1-xi)*(1-eta)
   phi2 = 0.25*(1+xi)*(1-eta)
   phi3 = 0.25*(1-xi)*(1+eta)
   phi4 = 0.25*(1-xi)*(1+eta)
   xd = phi1*x1+phi2*x2+phi3*x3+phi4*x4
   yd = phi1*y1+phi2*y2+phi3*y3+phi4*y4
   zd = phi1*z1+phi2*z2+phi3*z3+phi4*z4

penetr=-((xs-xd)*norm(1,ipair) + (ys-yd)*norm(2,ipair) +
            (zs-zd)*norm(3,ipair) - thick )
massd = (mass(1,nn1)*phi1+mass(1,nn2)*phi2+mass(1,nn3)*phi3 +
         mass(1,nn4)*phi4)/phivar2

    do 10 i=1,3
      ford(i)=massd*((phi1/mass(1,nn1))*(fore(i,nn1)-fori(i,nn1)) +
                      (phi2/mass(1,nn2))*(fore(i,nn2)-fori(i,nn2)) +
                      (phi3/mass(1,nn3))*(fore(i,nn3)-fori(i,nn3)) +
    10 continue
Veld(i) = phi1*Vold(i,mn1) + phi2*Vold(i,mn2) + phi3*Vold(i,mn3) + phi4*Vold(i,mn4)
dVel(i) = Veld(i) - Vold(i,islave)

continue
fornd = ford(1)*norm(1,ipair) + ford(2)*norm(2,ipair) +
for3 = norm(3,ipair)
masss = mass(1,islave)
for3n = (fore(1,islave)-fori(1,islave))*norm1,ipair1) +
       (fore(2,islave)-fori(2,islave))*norm2,ipair2) +
       (fore(3,islave)-fori(3,islave))*norm3,ipair3)
dvel = dVel(i)*norm(i,ipair) +
dVel(2)*norm(2,ipair) +
dVel(3)*norm(3,ipair)

Calculate normal contact force.

forcs = ((masss+massd)/(masss+massd))*(fornd/masss -
         forsn/masss + dveln/dtnew + penetr/(dtnew*dtnew))

Apply damping force if necessary

if (damping.ne.0) then
   forcs = forcs + damping*dveln
endif

if (forcs.lt.0.) forcs=0.
fori(1,islave) = forcs*norm(1,ipair)
for(2,islave) = forcs*norm(2,ipair)
for(3,islave) = forcs*norm(3,ipair)

Calculate friction forces if necessary.

if (fric.ne.0.) then
   do 20 i=1,3
      dVtan(i) = dVel(i) - dveln*norm(i,ipair)
      Fstan(i) = (fore(i,islave)-fori(i,islave)) -
                 forsn*norm(i,ipair)
      Fdtan(i) = ford(i) - fornd*norm(i,ipair)
   continue
   dvel = sqrt(dVtan(1)*dVtan(1) + dVtan(2)*dVtan(2) +
                dVtan(3)*dVtan(3))
   forst = sqrt(Fstan(1)*Fstan(1) + Fstan(2)*Fstan(2) +
               Fstan(3)*Fstan(3))
   forst = sqrt(Fdtan(1)*Fdtan(1) + Fdtan(2)*Fdtan(2) +
               Fdtan(3)*Fdtan(3))
   Ffric = ((masss+massd)/(masss+massd))* (forst/masss -
             forst/masss + dvel/dtnew)
check if already sliding

inode=invslave(islave)
if (slipflag(inode).eq.0) then
  Ffricmax = sfric*forsn
else
  Ffricmax = dfric*forsn
endif

limit friction force and project in sliding direction.

if (Ffric.gt.Ffricmax) then
  Ffric = dfric*forsn
  slipflag(inode)=1
else
  slipflag(inode)=0
endif
if (dvelt.ne.0) then
  tan(1) = dWtan(1)/dvelt
  tan(2) = dWtan(2)/dvelt
  tan(3) = dWtan(3)/dvelt
  forcc(1,islave) = forcc(1,islave) - Ffric*tan(1)
  forcc(2,islave) = forcc(2,islave) - Ffric*tan(2)
  forcc(3,islave) = forcc(3,islave) - Ffric*tan(3)
endif

Distribute the contact force over the master nodes

if (forces.ne.0.) then
  forcd = -forces
  Bigphi1 = (mass(1,mn1)*phi1)/(massd*phibar2)
  Bigphi2 = (mass(1,mn2)*phi2)/(massd*phibar2)
  Bigphi3 = (mass(1,mn3)*phi3)/(massd*phibar2)
  Bigphi4 = (mass(1,mn4)*phi4)/(massd*phibar2)
if ((sfric.ne.0).and.(dvelt.ne.0)) then
  forcedx = forcd*norm(1,ipair) + Ffric*tan(1)
  forcedy = forcd*norm(2,ipair) + Ffric*tan(2)
  forcedz = forcd*norm(3,ipair) + Ffric*tan(3)
else
  forcedx = forcd*norm(1,ipair)
  fordedy = forcd*norm(2,ipair)
  forcedz = forcd*norm(3,ipair)
endif
forcc(1,mn1) = forcc(1,mn1) + Bigphi1*forcedx
forcc(2,mn1) = forcc(2,mn1) + Bigphi1*forcedy
forcc(3,mn1) = forcc(3,mn1) + Bigphi1*forcedz
forcc(1,mn2) = forcc(1,mn2) + Bigphi2*forcedx

62
forc(2,mn2) = forc(2,mn2) + Bigphi2*forcedy
forc(3,mn2) = forc(3,mn2) + Bigphi2*forcedz
forc(1,mn3) = forc(1,mn3) + Bigphi3*forcedx
forc(2,mn3) = forc(2,mn3) + Bigphi3*forcedy
forc(3,mn3) = forc(3,mn3) + Bigphi3*forcedz
forc(1,mn4) = forc(1,mn4) + Bigphi4*forcedx
forc(2,mn4) = forc(2,mn4) + Bigphi4*forcedy
forc(3,mn4) = forc(3,mn4) + Bigphi4*forcedz
endif
100 continue
c
    return
end
Appendix B

Input files test problems

B.1 contact1.i

TITLE='Two plates in quasi-static contact'
!
A_DIR
    ANALYSIS=NONLINEAR  P_INT=1 1 1
    LCA=1  PAS=1.
    LCL=0  NCUT=5  NFACT=10  NSTRAT=0  MAXIT=6  MAXST=1
    EPSDIS=0.015  EPSR=0.02  ORFAC=1.00
END
!
! dynamic control parameters
!
DYNA
    TIME_END 0.002
    NPLOTS 20
    SCALEFACT 0.80
    F 1  1e-9  0.0  1e-3  1.0  .4  1.0  .49  1.0
    SHEAR_CORRECTION 0.8
    HMEMBRANE 0.04
    HGBEND 0.04
    HGSHEAR=0.04
    TIME_CALC
    END
!
! branch directives
!
BRANCH=1
B_DIR
    MATERIAL=ELASTIC DEFORM=NONLINEAR
END
!
! element definition
ELEM
TYPE=Q4.ETA MID=1
NG=7 NINTZ=2
THICK=1.0
  1 1 2 3 4
  2 5 6 7 8
END

! node definition
!
NODENES
  1 0.0 0.0 0.0
  2 10. 0.0 0.0
  3 10. 0.0 10.
  4 0.0 0.0 10.
  5 2.0 0.0 2.0
  6 8.0 0.0 2.0
  7 8.0 0.0 8.0
  8 2.0 0.0 8.0
END

! contact definition

CONTACT
  CSID 1
  CTYPE 1
  DAMP 0.004
  DIRFLAG -1
  MASTER 1 1 2 3 4
  SLAVE 5/8
ENDC

! material definition

EMATERIALS
  MID=1 TYPE=ISU  E=99408.0
  P=0.3
  DENS=2.6477e-9
ENDMID

END

! prescribed velocities
!
PRESEVEL
  V=0.0
  N 1.0 0.0 0.0
  TRAN 1/8 ROT 1/8
N 0.0 1.0 0.0
TRAN 1/4 ROT 1/8
N 0.0 0.0 1.0
TRAN 1/8 ROT 1/8
END
!
! nodal directors
!
DIRC
D 0.0 -1.0 0.0
END
!
! external forces
!
FORCES
CASE=1
TYPE=FORCE
DOF=2 p=-1.0 5/8
END
!
END_BRANCH
RUN
B.2 contact2.i

TITLE='Two plates hitting'
!
A_DIR
   ANALYSIS=NONLINEAR P_INT=1 1 1
   LCA=1  PAS=1.
   LCL=0 NCUT=5 NFACT=10 NSTRAT=0 MAXIT=6 MAXST=1
   EPSDIS=0.015 EPSR=0.02 DRFAC=1.00
END
!
Dyna
TIME_END 0.002
NPLTS 20
SCALEFACT 0.80
F 1 1e-9 0.0 1e-3 1.0  .4 1.0 .49 1.0
SHEAR_CORRECTION 0.8
HOMEMBRANE 0.04
HGBEND 0.04
HGSHEAR=0.04
TIME_CALC
END
!
BRANCH=1
B_DIR
   MATERIAL=ELASTIC DEFORM=NONLINEAR
END
!
ELEM
   TYPE=Q4.ETA MID=1
   NQ=7 NINTZ=2
   THICK=1.0
       1 1 2 3 4
       2 5 6 7 8
END

NODES
   1 0.0  0.0  0.0
   2 10.0  0.0  0.0
   3 10.0  0.0  10.0
   4 0.0   0.0  10.0
   5 2.0  -1.0  2.0
   6 8.0  -1.0  2.0
   7 8.0  -1.0  8.0
   8 2.0  -1.0  8.0
END
!
CONTACT
   CSID 1
   CTYP 1
DAMP 0.004
DIRFLAG -1
MASTER 1 1 2 3 4
SLAVE 5/8
ENDC
END

! EMATERIALS
MID=1 TYPE=ISO  
  E=99408.0
  P=0.3
  DENS=2.6477e-9
END

ENDMID
END

! INITVEL
  V=1000.
  N 0.0 1.0 0.0
  TRAN 5/8
END

! PRESVEL
  V=0.0
  N 1.0 0.0 0.0
  TRAN 1/8 ROT 1/8
  N 0.0 1.0 0.0
  TRAN 1/4 ROT 1/8
  N 0.0 0.0 1.0
  TRAN 1/8 ROT 1/8
END

! DIRC
  D 0.0 -1.0 0.0
  ALL
END

! dummy forces
!
FORCES
  CASE=1
  TYPE=FORCE
  DOF=2 p=0.0 5/8
END

END_BRANCH
RUN
B.3 friction.i

TITLE='Contacting plates with friction'
!
A_DIR
  ANALYSIS=NONLINEAR  P_INT=1 1 1
  LOCA=1  PAS=1.
  LCL=0 NCUT=5 NFACT=10 NSTRAT=0 MAXIT=6 MAXST=1
  EPSDIS=0.015 EPSR=0.02 ORFAC=1.00
END
!
Dyna
  TIME_END 0.0015
  NPLTS 20
  SCALEFACT 0.80
  F 1 1e-9 0.0 1e-6 1.0 .4 1.0 .49 1.0
  F 2 1e-9 0.0 5e-4 1.0 1e-3 0.0 .49 0.0
  SHEAR_CORRECTION 0.8
  HMEMBRANE 0.04
  HGBEND 0.04
  HGSHEAR=0.04
  TIME_CALC
  END
!
BRANCH=1
B_DIR
  MATERIAL=ELASTIC DEFORM=NONLINEAR
END
!
ELEM
  TYPE=Q4.ETA MID=1
  NG=7 NINTZ=2
  THICK=1.0
    1 1 2 6 5
    2 2 3 7 6
    3 3 4 8 7
    4 9 10 11 12
END
!
NODES
  1 0.00 0.00 0.00
  2 10.0 0.00 0.00
  3 20.0 0.00 0.00
  4 30.0 0.00 0.00
  5 0.00 10.0 0.00
  6 10.0 10.0 0.00
  7 20.0 10.0 0.00
  8 30.0 10.0 0.00
!
9 25.0 2.00 0.01
10 29.0 2.00 0.01
11 29.0 8.00 0.01
12 25.0 8.00 0.01
END
!
CONTACT
  CSID 1 CTYPE 1
    SFRIC 0.4
    DFRIC 0.3
  MASTER
    1 1 2 6 5
    2 2 3 7 6
    3 3 4 8 7
  SLAVE
    9/12
ENDC
END
!
EMATERIALS
  MID=1 TYPE=ISO  E=99408.0
    P=0.3
    DENS=2.6477e-9
ENDMID
END
!
INITVEL
  V=0.0
  N 0.0 0.0 1.0
  TRAN 9/12
END
!
PRESVEL
  V=0.0
  N 1.0 0.0 0.0
  TRAN 1/8 ROT 1/12
  N 0.0 1.0 0.0
  TRAN 1/12 ROT 1/12
  N 0.0 0.0 1.0
  TRAN 1/8 ROT 1/12
END
!
DIRC
  D 0.0 0.0 1.0
  ALL
END
!
FORCES
  CASE=1
TYPE=FORCE
DOF=3  P=-20.  9/12
CASE=2
TYPE=FORCE
DOF=1  P=-10.  9/12
END
!
END_BRANCH
RUN
B.4 beam.i

The ETA input file of the beam against the rigid wall

TITLE='beam on rigid wall'

DYNA
    TIME_END 0.0025
    NPLOTS 50
    SCAEFACT 0.8
    F1 1E-9 1.0 0.3 1.0 0.4 1.0 0.5 1.0
    SHEAR_CORRECTION 0.8
    HGMEMBRANE 0.04
    HGBEND 0.04
    HGSHEAR 0.04
    TIME_CALC
END
!
BRANCH=1

B_DIR
    MATERIAL=ELASTIC DEFORM=NONLINEAR
END

ELEM
    TYPE=Q4.ETA MID=1
    NG=1 NINTZ=4
    THICK=0.1
  1  1  2  23  22
  2  2  3  24  23
  3  3  4  25  24
  4  4  5  26  25
  5  5  6  27  26
  6  6  7  28  27
  7  7  8  29  28
  8  8  9  30  29
  9  9 10  31  30
 10 10 11  32  31
 11 11 12  33  32
 12 12 13  34  33
 13 13 14  35  34
 14 14 15  36  35
 15 15 16  37  36
 16 16 17  38  37
 17 17 18  39  38
 18 18 19  40  39
 19 19 20  41  40
 20 20 21  42  41

TYPE=Q4.ETA MID=1
    NG=1 NINTZ=6
    THICK=1.0
  1  43  44  45  46

END
<table>
<thead>
<tr>
<th>NODES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>17</td>
</tr>
<tr>
<td>18</td>
</tr>
<tr>
<td>19</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>21</td>
</tr>
<tr>
<td>22</td>
</tr>
<tr>
<td>23</td>
</tr>
<tr>
<td>24</td>
</tr>
<tr>
<td>25</td>
</tr>
<tr>
<td>26</td>
</tr>
<tr>
<td>27</td>
</tr>
<tr>
<td>28</td>
</tr>
<tr>
<td>29</td>
</tr>
<tr>
<td>30</td>
</tr>
<tr>
<td>31</td>
</tr>
<tr>
<td>32</td>
</tr>
<tr>
<td>33</td>
</tr>
<tr>
<td>34</td>
</tr>
<tr>
<td>35</td>
</tr>
<tr>
<td>36</td>
</tr>
<tr>
<td>37</td>
</tr>
<tr>
<td>38</td>
</tr>
<tr>
<td>39</td>
</tr>
<tr>
<td>40</td>
</tr>
<tr>
<td>41</td>
</tr>
<tr>
<td>42</td>
</tr>
<tr>
<td>43</td>
</tr>
<tr>
<td>44</td>
</tr>
<tr>
<td>45</td>
</tr>
<tr>
<td>46</td>
</tr>
</tbody>
</table>

END
EMATERIALS
MID=1  TYPE=ISO  E=1.0e6
P=0.0
DENS=0.01
ENDMID
END
CONTACT
CSID=1
CTYPE=1
GSEARCH 1
MASTER 1 43 44 45 46
SLAVE 21/42/21
ENDC
END
INITVELO
V=1.0
N 1.0 0.0 0.0
TRAN 1/42
END
PRESLVELO
V=0.0
N 1.0 0.0 0.0
TRAN 43/46 ROT 43/46
N 0.0 1.0 0.0
TRAN 43/46 ROT 43/46
N 0.0 0.0 1.0
TRAN 43/46 ROT 43/46
END
DIRC
D 0.0 0.0 1.0
1/42
D 1.0 0.0 0.0
43/46
END
FORCES
CASE=1
TYPE=FORCE
DOF=2 p=0.0 3
END
!
END_BRANCH
RUN
B.5 beam.inp

The ABAQUS input file of the beam against the rigid wall

*HEADING
Cantilever beam against rigid wall
**
*RESTART, WRITE, FREQ=1
**
** define beam
**
*NODE, NSET=ENDS
  1,0.
  21,10.
*GEN, NSET=BALKNOD
  1,21
ELEMENT, TYPE=B21
  1,1,2
*ELGEN, ELSET=BALK
  1,20
BEAM SECTION, SECT=RECTANGULAR, ELSET=BALK, MATERIAL=TEST
  1.,0.1
*MATERIAL, NAME=TEST
  ELASTIC
  1.E6,0.
  DENSITY
  0.01
**
** define rigid wall
**
*NODE
  30,10.,-1.
  31,10.,1.
*ELEMENT, TYPE=R2D2, ELSET=FOUND
  30,30,31
*RIGID BODY, ELSET=FOUND, REF NODE=30
  BOUNDARY
  30, ENCASTRE
*ELSET, ELSET=ELLAST
  20
*NSSET, NSET=NLAST
  21
*SURFACE DEFINITION, NAME=RGDFOUND
  FOUND, SPOS
*SURFACE DEFINITION, NAME=BEAM
  ELLAST, SPOS
*CONTACT PAIR, INTERACTION=NO_FRI, SMALL SLIDING
  BEAM, RGDFOUND
*SURFACE INTERACTION, NAME=NO_FRI
  0.0
*INITIAL CONDITIONS, TYPE= VELOCITY
BALKNOD, 1, 1.
**
*NSET, NSET=P21
21
*NSET, NSET=P30
30
**
*STEP, INC=100, NLGEOM
**
*DYNAMIC, HAFTOL=1.E10
0.00005, 0.0025, 0.00005
**
*NODE PRINT, NSET=P21, FREQUENCY=1, SUM=NO
  U1
*NODE PRINT, NSET=P30, FREQUENCY=1, SUM=NO
  RFI
*EL PRINT, FREQ=0
**
*ENDSTEP
Contents

1 Introduction ................................................. 3

2 The ETA Command Language ......................... 5
   2.1 dynamic ................................................. 7
   2.2 elements ............................................... 9
   2.3 nodes ................................................ 11
   2.4 ematerials ............................................. 12
   2.5 initvelocities ....................................... 14
   2.6 presvelocities ...................................... 15
   2.7 juncures .............................................. 16
   2.8 contact ............................................... 18
   2.9 dirnormals .......................................... 21
   2.10 forces ................................................ 22

3 The PCL Commands ..................................... 23

4 Example .................................................. 25

Bibliography .............................................. 29

Index ....................................................... 31
Chapter 1

Introduction

This document describes the specific command syntax for the input file for the B2000 macro processor ETA (Explicit Transient Analysis). As ETA is based on B2000, it uses the same syntax for most commands as the B2000 macro-processors [3, 6], but it features some new commands for the dynamic behaviour of the model as well as specific ETA-defined commands, like the ETA-type junctures. The input file is processed by the input processor (referred to as IP) and written to a database by the MEM-COM data management system [4, 5].

Apart from that, ETA can be seen as an independent finite element code, using an explicit time integration scheme to integrate the solution in time. Because of this scheme, ETA can only be used for dynamic problems. The first version of the finite element code ETA (version 1.1) was written in 1994 by Rogier Fransen [1, 2]. This version featured only the basic tools necessary: the control module, the explicit time integration, a shell element and junctures. After put to rest for 2 years development was continued with the introduction of version 1.2 which included plasticity [8], contact [7] and computational restart.

Based on personal experience working with B2000/ETA this report is written with the intent to facilitate writing input files for (beginning) users. Because the current version of ETA is written as part of two master theses, the command syntax information is to be found in different reports. This document is meant to collect the specific input commands and explain their syntax the same way this can be found for B2000 in [3, 6]. PCL commands in Chapter 3 are the commands that can be given at the command prompt of the ETA program. In Chapter 4 an example of an input file for ETA is given and explained.
Chapter 2

The ETA Command Language

The ETA finite element code uses the same data structure and input syntax as the B2000 macro processors. This chapter explains only the specific ETA commands, others can be found in the B2000 Processors Reference Manual [6] To analyse a dynamic problem with ETA the input file must be compiled and written to a database by the IP before it can be processed by ETA. The following commands run the input processor to process the input file demo.i to the database file demo.adb:

```
\% b2ipeta

B2000 Input processor (B2IP), version 1.8

B2IP> adb demo.adb  # Attach the database demo.adb
B2IP> input demo.i  # Read the input file demo.i
B2IP> go            # Start the IP
```

Then the database demo.adb can be used by ETA. The computations are started by:

```
\% b2eta

B2000 Explicit Transient Analysis Processor (B2ETA), version 1.2

Command: adb demo.adb  # Attach the database demo.adb
Command: go             # Start ETA
```

The input file demo.i will schematically look like:

ETA User’s Manual 5
title='demo'
!
analysis directives
branch 1
elem
element definition
node
node definition
emat
material definition
boundary conditions
initial conditions
etc...
endbranch
go

NOTICE that ETA contains only one branch.
2.1 dynamic

STATUS: required

The DYNAMIC command defines the parameters for dynamic analysis. No default values are assumed.

Synopsis

dyna
  parameters
end

Parameters

- **time_start ts**
  Specifies the starttime of the simulation.

- **time_end te**
  Specifies the endtime of the simulation.

- **scalefactor s**
  Specifies the scale factor for the stable time step. The critical time step is calculated and multiplied by this factor to obtain the time step used.

- **nplots np**
  Specifies the number of output plots between the starttime and the endtime.

- **shear_correction s**
  Specifies the shear correction factor. Default is 0.8.

- **hgmembrane hm**
  Specifies the hourglass stiffness ratio for membrane behaviour. Values between 0.01 and 0.05 are recommended.

- **hgbending hb**
  Specifies the hourglass stiffness ratio for bending behaviour. Values between 0.01 and 0.05 are recommended.

- **hgshear hs**
  Specifies the hourglass stiffness ratio for out-of-plane shear behaviour. Values between 0.01 and 0.05 are recommended.
2.1 dynamic

- *time_calculation*
  Flag to indicate that the time step is adjusted during the calculation.

- *function lcid t1 f1 t2 f2 t3 f3 t4 f4*
  Specifies the load weighting function in time for load case lcid.

**Examples**

Set dynamic control variables:

```
Dyna
  TIME_END  0.003
  NPLOTS 20
  SCALEFACT  0.80
  F 1  1e-9  0.0  1e-4  0.5  1E-3  1.0  .49  1.0
  SHEAR_CORRECTION  0.8
  HGMEMBRANE  0.04
  HGBEND  0.04
  HGSHREAR  0.04
  TIME_CALC
END
```
2.2 elements

STATUS: required

The ELEMENT command generates the element by defining geometry of the element. In ETA there is only one shell element available, so that the needed element parameters are limited.

Synopsis

elem
  parameters
    element connectivity list
  parameters
    element connectivity list
end

Parameters

• \textit{type etype}
  Defines the element type. Only a 4 node quadrilateral shell element is available: Q4.ETA.

• \textit{mid imat}
  Defines the material definition number of the element.

• \textit{ng n}
  Defines the number of integration points over the surface. For element Q4.ETA this is (internally) set to 1.

• \textit{nintz n}
  Defines the number of integration points through the thickness

• \textit{thick t}
  Defines the thickness for the elements.

• \textit{iel, n1, n2, n3, n4}
  Defines the external element number \texttt{iel} and its node list. Different sets of elements may have their own external element numbers, but B2000/ETA uses only the internal element numbers.
Examples
Define a shell element of thickness 1.0 and 2 integration points:

ELEM
    TYPE=Q4.ETA MID=1
    NG=7 NINTZ=2
    THICK=1.0
        1 1 2 3 4
END
2.3 nodes

STATUS: required

The command NODES generates the nodal coordinates in the global coordinate system.

Synopsis

nodes
  parameters
end

Parameters

- \textit{inode, x, y, z}
  Defines the cartesian coordinates $x$, $y$, $z$ for node number \texttt{inode}

Examples

Define 4 nodes to create a rectangle in the x-y plane:

\begin{verbatim}
NDOEs
  1  0.0  0.0  0.0
  2  15.0  0.0  0.0
  3  15.0  10.0  0.0
  4   0.0  10.0  0.0
END
\end{verbatim}
2.4 ematerials

STATUS: required

The EMAT command defines the material properties assigned to the elements. The elements refer to the corresponding material properties by the material ID number mid. Although B2000 can use all kinds of materials, ETA supports only two types: the general isotropic elastic material and the fraction model for isotropic plasticity.

Materials are defined constitutively from 1 to the number of element. Gaps in the numbering must be avoided.

Synopsis

emat
  mid parameters
  endmid
  mid parameters
  endmid
  end

Parameters

- *mid imat*
  Defines the material identification number. All subsequent parameters refer to the material number imat until endmid is specified. Material identification numbers must be numbered constitutively 1, 2, 3...

- *type mtype*
  Refers to the material type. ETA supports only isotropic material ISO.

- *e val*
  Specifies Young’s modulus.

- *g val*
  Specifies the shear modulus.

- *p val*
  Specifies Poisson’s coefficient and should be in the range of 0.0 to 0.5.

- *dens val*
  Specifies the density of the material. Although optional in B2000, ETA requires the density, as the nodal masses must be calculated.
• *ptty ptype*  
  Specifies the plasticity type. ETA supports only the fraction model  
  FRAC with the following parameters:

  - *iter / not*  
    Flag indicating whether iteration must be performed on the fraction  
    data or not.

  - *nfrac nf*  
    Number of fractions. The number of fractions must be one less  
    than the number of breakpoints in the stress-strain curve.

  - *nsubint n*  
    Number of subincrements. Not supported.

  - *epsX val*  
    Strain at the breakpoint of the one-dimensional stress-strain curve.  
    X goes from 1 to *nf-1*.

  - *sigX val*  
    Stress at the breakpoint of the one-dimensional stress-strain curve.  
    X goes from 1 to *nf-1*

• *endmid*  
  Terminates the material definition of the current material.

**Examples**

Some isotropic material, plasticity data commented out:

```
EMATERIALS
  MID=1 TYPE=ISO  E=99408.0
  P=0.3
  DENS=2.6477e-9
!  PLTY FRAC ITER
!  NFRAC 2
!  NSUBINT 1
!  EPS1 0.0012 SIG1 120.0
!  EPS2 0.0023 SIG2 185.0
!  EPS3 0.0033 SIG3 230.0
  ENDMID
END
```

ETA User's Manual 13
2.5 initvelocities

STATUS: optional

The INITVEL command sets the initial conditions in the form of initial velocities. The initial velocities can be either translational or rotational velocities. Multiple initial velocities can be given for the same node and are vectorially added.

Synopsis

init
  parameters
  end

Parameters

- velocity \( v \)
  Sets the value of the initial velocity

- normal \( nx, ny, nz \)
  Sets the direction of the initial velocity. The vector is normalized to unit length.

- tran \( n1/n2/n3 \)
  Sets translational velocities for node list from \( n1 \) to \( n2 \) in steps of \( n3 \).

- rot \( n1/n2/n3 \)
  Sets rotational velocities for node list from \( n1 \) to \( n2 \) in steps of \( n3 \).

Examples

Set initial velocities for nodes 5, 7 and 9:

INITVEL
  \( V=1000 \).
  N 0.0 1.0 0.0
  TRAN 5/9/2
END
2.6 presvelocities

STATUS: optional

The PRESVEL command sets the boundary conditions in the form of pre-
scribed velocities. The prescribed velocities can be either translational or
rotational velocities. Multiple velocities can be given for the same node and
are vectorially added only if they do not conflict.

Synopsis

pres
  parameters
  end

Parameters

- *velocity v*
  Sets the value of the initial velocity

- *normal nx, ny, nz*
  Sets the direction of the initial velocity. The vector is normalized to
  unit length.

- *tran n1/n2/n3*
  Sets translational velocities for node list from n1 to n2 in steps of n3.

- *rot n1/n2/n3*
  Sets rotational velocities for node list from n1 to n2 in steps of n3.

Examples

Set prescribed velocities for nodes 5, 7 and 9:

PRESVEL
  V=0.0
  N 0.0 1.0 0.0
  TRAN 5/9/2
END
2.7 juncures

STATUS: optional

The ETAJUNC command defines the juncures for ETA. Juncures connect one or more translational degrees of freedom and the rotational degrees of freedom of two separate nodes. For a more detailed explanation refer to [2], section B-7.2.

Synopsis

etaj
  parameters
endjunc
end

Parameters

- **type type**
  Defines the type of juncture. There are 5 juncture types:
  Type 1 connects the rotational degrees of freedom for two or more nodes with different fiber direction vectors.
  Type 2 connects the rotational degrees of freedom for two or more nodes with the same fiber direction vector.
  Type 11 couples the translational degrees of freedom of two or more nodes in one spatial direction.
  Type 12 couples the translational degrees of freedom of two or more nodes in two spatial directions.
  Type 13 couples the translational degrees of freedom of two or more nodes in all directions.

- **nodes n1, n2, ..., nn**
  Specifies the nodes to be joined.

- **N1 nx, ny, nz**
  Specifies the first direction in which the degrees of freedom are coupled. Only necessary for type 11 and 12.

- **N2 nx, ny, nz**
  Specifies the first direction in which the degrees of freedom are coupled. Only necessary for type 12.
Examples

Couple node 1 with node 2 and node 3 with node 4 in all directions. The coupled nodes have the same fiber vectors:

ETAJUNC
  TYPE=13
  NODES  1  2
  ENDJUNC
  NODES  3  4
  ENDJUNC
  TYPE=2
  NODES  1  2
  ENDJUNC
  NODES  3  4
  ENDJUNC
END
2.8 contact

STATUS: optional

The CONTACT command defines the contact interfaces. It features a node-to-segment contact, requiring the definition of the contacting master segment (not necessary the same as a structural element) and the potential contacting slave nodes.

Synopsis

cont
  parameters
endc
end

Parameters

- **csid id**
  Contact surface ID number. The first defined surface should be numbered 1, etc.

- **ctype type**
  Contact type. ETA currently features three different contact types, although contact type 3 is not to be used.
  **Type 1** is a fully elastic node-to-segment contact, requiring the slave and master nodes to be part of a previous defined element.
  **Type 2** is a rigid node-to-segment contact, requiring only the slave nodes to be part of a previous defined element. The nodes of the master segment must have prescribed velocities in all degrees of freedom.
  **Type 3** is a node-to-segment contact, but it uses the penalty method to calculate the normal contact force. Friction is not included and the penalty parameter is fixed. *This contact type is not to be used.* It is implemented for research purpose only.

- **thick, mthick mt**
  Defines the contact thickness of the master segment. The default value is 0.0

- **sthick st**
  Defines the contact thickness of the slave segment. The default value is 0.0
• **damping d**
  Defines the contact damping factor. Unfortunately this must be the real damping factor instead of a ratio of the critical damping. The default value is 0.0

• **friction, sfiction sf**
  Defines the static friction coefficient. The default value is 0.0

• **dfraction df**
  Defines the dynamic friction coefficient. If only sf is defined, then df equals sf. Default value is 0.0

• **gssearch gs**
  Global search counter. Global search is performed every sg cycles. Default value is 21.

• **dirflag dir**
  Flag to define outward direction. Default takes the master segment nodes counter-clockwise. dir -1 sets the other side of the segment as outward. Default is 1.

• **master n1, n2, n3, n4, n5**
  Defines master segment number n1 by nodes n2 to n5. Only internal segment numbers are stored.

• **slave n1/n2/n3**
  Defines slave node list to the contact surface.

• **endc**
  Terminates the contact surface definition of the current surface.

**Examples**

Define contact surface by two master segments of nodes 1 to 6 and slave nodes 8 to 12. The shell thickness of the master segment is 1.0 and the friction coefficient is 0.3. Notice that the contact thickness is half the shell thickness.

```
CONTACT
  CSID 1
  CTYP 1
  MTHICK 0.5
  FRICTION 0.3
```
2.8 contact

MASTER
  1 1 2 4 3
  2 3 4 6 5
SLAVE
  8/12
ENDC
END
2.9 dirnormals

STATUS: required for element type Q4.ETA

The DIRN command defines the nodal direction (fiber) vectors needed for the shell element Q4.ETA.

Synopsis

dirn
    parameters
    end

Parameters

- **director nx, ny, nz**
  
  Defines the direction of the nodal fiber direction vector. The vector is normalized to unit length.

- **nodelist n1/n2/n3, all**
  
  Specifies the node list for which the fibers are defined. All assigns the specified direction vector to all nodes. If a fiber direction vector is defined more than once for the same node only the last definition is taken.

Examples

Set direction vector in y-direction for all nodes:

DIRC
    D 0.0 1.0 0.0
    ALL
    END
2.10 forces

STATUS: required

Due to a bug in the ETA control module an external force must be given. The FORCE command defines mechanical forces on nodes. Forces are formulated in the global coordinate system.

Synopsis

forces
  parameters
  end

Parameters

- case lcid
  Specifies the load case number. All subsequent forces are assigned to the load case until another case is entered. At most two cases may be applied.

- dof id
  Specifies the nodal degree of freedom for which the force is applied.

- p val
  Specifies the force magnitude.

- n1/n2/n3
  Node list to which the value of p will be assigned.

Examples

Apply dummy forces to some nodes:

FORCES
  CASE=1
  TYPE=FORCE
  DOF=2 p=0.0 5/8
END

22 ETA User’s Manual
Chapter 3

The PCL Commands

The Processor Command Language (PCL) commands change some variables in the database before the calculations start, without reprocessing the input file. They can be issued after starting ETA to override values in the database. ETA supports the following PCL commands:

- `time_start ts`
  Specifies the start time of the simulation.

- `time_end te`
  Specifies the end time of the simulation.

- `scalefactor s`
  Specifies the stable time step scalefactor.

- `nplots np`
  Specifies the number of output plots between the start time and the end time of the total simulation.

- `restart nr`
  Flag indicating restart from output plot nr. Note tat the number of output plots defined by `nplots` is the total number of output plots, thus the number of output plots after restart becomes `np-nr`.

The PCL commands may be abbreviated if possible.
Chapter 4

Example

If we go through Chapter 2 step by step we are able to write an input file for ETA. If we take a simple model of a strip, clamped on one side and hit by a small shell as shown in Fig.(4.1). We take a strip of length 30, width 10 and thickness 1 and model it with 3 shell elements. The impacting shell is made of 1 element. The impacting shell has an initial velocity of 1000 downward. Assuming that the impacting shell can only come in contact with the two elements at the end of the strip, we work our way through all the necessary input commands, obtaining the following input file:

![Diagram](image)

Figure 4.1: Example model
TITLE='demo input file'
!
! dynamic control parameters
!
Dyna
  TIME_END 0.003
  NPLOTS 20
  SCALEFACT 0.80
  F 1 1e-9 0.0 1e-4 1.0 .4 1.0 .49 1.0
  SHEAR_CORRECTION 0.8
  HGMEMBRANE 0.04
  HGBEND 0.04
  HGSHEAR=0.04
  TIME_CALC
END
!
! branch directives
!
BRANCH=1
B_DIR
  MATERIAL=ELASTIC DEFORM=NONLINEAR
END
!
! element definition
!
ELEM
  TYPE=Q4.ETA MID=1
  NG=7 NINTZ=3
  THICK=1.0
  1 1 2 6 5
  2 2 3 7 6
  3 3 4 8 7
  TYPE=Q4.ETA MID=2
  NG=7 NINTZ=1
  THICK=1.0
  1 9 10 12 11
END
!
! nodal coordinates
!
NODES

26 ETA User's Manual
1 0.0  -5.0  0.0
2 10.0  -5.0  0.0
3 20.0  -5.0  0.0
4 30.0  -5.0  0.0
5 0.0  5.0  0.0
6 10.0  5.0  0.0
7 20.0  5.0  0.0
8 30.0  5.0  0.0
9 24.0  -2.0  1.0
10 28.0  -2.0  1.0
11 24.0  2.0  1.0
12 28.0  2.0  1.0

END

! material definitions

EMATERIALS
   MID=1 TYPE=ISO   E=71000.0
         P=0.3
         DENS=2.6477e-9

ENDMID

MID=2 TYPE=ISO   E=200000.0
         P=0.3
         DENS=8.0e-9

ENDMID

END

! initial velocities

INITVEL
   V=-1000.
   N 0.0 0.0 1.0
   TRAN 9/12

END

! prescribed velocities

PRESVEL
   V=0.0
   N 1.0 0.0 0.0
   TRAN 1/5/4 ROT 1/5/4
N 0.0 1.0 0.0
TRAN 1/5/4 ROT 1/5/4
N 0.0 0.0 1.0
TRAN 1/5/4 ROT 1/5/4
END
!
! contact surface definition
!
CONTACT
CSID 1
CTYP 1
MASTER
   1 2 3 7 6
   2 3 4 8 7
SLAVE
   9/12
ENDC
END
!
! nodal dirctors
!
DIRC
   D 0.0 0.0 1.0
   ALL
   END
!
! external (dummy) forces
!
FORCES
   CASE=1
   TYPE=FORCE
   DOF=2 p=0.0 5/6
END
!
END_BRANCH
RUN
BIBLIOGRAPHY

Bibliography


Index

contact, CONT, 18
dynamic, DYNA, 7
elements, ELEM, 9
end time, 7, 23
example, 25
forces, FORCE, 22
friction, 19
hourglass, 7
initial velocities, INIT, 14
junctures, ETAJ, 16
materials, EMAT, 12
nodes, NODE, 11
normals, DIRN, 21
PCL, 23
plasticity, 13
plots, 7, 23
prescribed velocities, PRES, 15
restart, 23
scalefactor, 7, 23
shear correction, 7
start time, 7, 23
time_end, 7, 23
time_start, 7, 23